Development of a Satellite Network Simulator Tool and Simulation of AX.25, FX.25 and a Hybrid Protocol for Nano-Satellite Communications

by

Jan-Hielke Le Roux

Thesis presented in fulfilment of the requirements for the degree of
Master of Engineering
at Stellenbosch University

Supervisors:

Mr A. Barnard Dr R. Wolhuter

Department Electrical and Electronic Engineering

December 2014

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the copyright thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

October 2014

Abstract

Nano-satellites are mostly used in lower earth orbit applications, where communication intervals are limited, often to a combined total of less than one hour per day. With these type of inherent limitations of lower earth orbits, there are also the physical size and equipment restriction of nano-satellites to consider, especially those of the CubeSat specification. It is of critical importance to use the limited time and communication resources as effectively as possible.

The network protocol has a huge influence on reliability and throughput of a satellite network. An important requisite for designing, comparing and improving network protocols is a network protocol simulator, that is able to envisage the design results. Simulation can facilitate rapid development and unforeseen discoveries. Very little information is currently available regarding communication protocols used in nano-satellites. This thesis aims to explore and improve the current status of nano-satellite network simulation, as well as to demonstrate the development of an improved communication protocol strategy.

It was found that there is a lack of proper simulation tools for satellite networks, which led to the development of SatSim. SatSim is a discrete event network simulation tool, developed in Python, which can be used to develop and analyse network protocols. SatSim was verified by comparing simulation results with other published results, which made use of different software tools and theoretical throughput calculations.

AX.25 is one of the most commonly used network protocols in the nano-satellite industry. It was implemented in SatSim and verified with theoretical throughput calculations, as no other simulation data on AX.25 was available. AX.25 was used as a baseline protocol to improve upon. FX.25 was developed by the Stensat Group in an attempt to improve AX.25. FX.25 adds forward error correction to AX.25, by wrapping additional data around the AX.25 frames. This method maintains backward compatibility with AX.25. FX.25 was implemented in SatSim and the simulation results proved that FX.25 was a more reliable protocol than AX.25, as it can communicate at lower elevations and over noisier communication channels. However, the drawback of the additional forward error correction is the increased overhead, which reduces the overall payload data throughput.

A modular AX/FX.25 protocol was then implemented in SatSim, to exploit the strengths of both protocols. This hybrid protocol yielded significant improvements to data throughput and can enable future software defined radio or hardware developments.

Opsomming

Nano-satelliete word hoofsaaklik gebruik in lae-aard wentelbaan toepassings waar kommunikasietyd beperk is, soms tot minder as een uur per dag. Gepaardgaande met hierdie inherente beperking van lae-aard wentelbane, is daar ook die verminderde omvang en kapasiteit van nano-satelliete, veral ten opsigte van die CubeSat spesifikasie. Effektiewe aanwending van die beperkte tyd en kommunikasie-hulpbronne is dus noodsaaklik.

Die keuse van netwerk protokol het 'n beduidende invloed op die betroubaarheid en data deurset van 'n satelliet netwerk. 'n Belangrike voorvereiste vir die ontwerp, vergelyking en verbetering van netwerk-protokolle, is 'n netwerk simulator. Beperkte inligting is tans beskikbaar oor kommunikasie protokolle in nano-satelliet toepassings. Hierdie tesis fokus op die verbetering van nano-satelliet netwerk-simulasie, asook die ontwikkelling van 'n verbeterde netwerk-protokol strategie vir nano-satelliet toepassings.

Dit het na vore gekom dat daar 'n leemte is in die beskikbaarheid van simulasie sagteware wat gerig is op die ondersoek van satelliet netwerke. Hierdie waarneming het die ontwikkeling van SatSim genoop. SatSim is 'n diskrete-gebeurtenis netwerk-simulasie sagtewarepakket wat in die Python programmeertaal ontwikkel is om netwerk protokolle te ontwikkel en te analiseer. SatSim was geverifieer deur simulasies te vergelyk met die resultate van ander navorsingspublikasies, wat van verskillende sagtewarepakkette gebruik gemaak het, sowel as teoretiese deursetberekeninge.

AX.25 is een van die netwerk protokolle wat mees algemeen in die nano-satelliet bedryf voorkom. AX.25 was geïmplementeer in SatSim en geverifieer met teoretiese deursetberekeninge. AX.25 was gebruik as 'n grondslag om op te verbeter. FX.25 was ontwikkel deur die Stensat Group in 'n poging om op AX.25 te verbeter. FX.25 voeg vorentoe-foutkorreksie by tot AX.25, deur addisionele data tot die AX.25 netwerk pakkies te voeg. Hierdie benadering bewerkstellig agteruit-verenigbaarheid met AX.25. FX.25 was geïmplementeer in SatSim en simulasieresultate dui daarop dat FX.25 'n meer betroubare protokol is as AX.25, omdat dit teen laer elevasiehoeke en oor swakker kommunikasiekanale kan kommunikeer. Die verbeterde betroubaarheid is ten koste van datadeurset, as gevolg van die toevoeging van die vorentoe-foutkorreksiedata.

'n Modulêre AX/FX.25 protokol was geïmplemeteer om te kapitaliseer op die sterk eienskappe van beide protokolle. Hierdie hibriede protokol het 'n beduidende verbetering gelewer ten opsigte van data deurset en kan toekomstige sagteware-gedefinieerde radio en hardeware-toepassings stimuleer.

Acknowledgements

I would like to extend my gratitude to the following people. Without them this work would not have been possible.

- Mr. Arno Barnard and Dr. Riaan Wolhuter, for their professional guidance, valued direction, support and patience.
- My parents, family and friends, for their endearing support and motivation.
- All the fellow lab-mates in the ESL, for their camaraderie and valued input.
- Soli Deo gloria.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations and Acronyms

ADCS	Attitude Determination and Control System
AGC	Automatic Gain Control
ASK	Amplitude Shift Keying
BER	Bit Error Rate
BFSK	Binary Frequency Shift Keying
CDMA	Code Division Multiple Access
COTS	Commercial Off-The-Shelf
CRC	Cyclical Redundancy Check
CTS	Clear To Send
ESL	Electronic Systems Laboratory
FEC	Forward Error Correction
FDMA	Frequency Division Multiple Access
FPGA	Field Programmable Gate Array
FSK	Frequency Shift Keying
GEO	Geostationary Earth Orbit
GUI	General User Interface
HDLC	High-Level Data Link Control
I/O	Input/Output
ISO	International Organisation for Standardisation
ISL	Inter Satellite Link
ISM	Industrial, Scientific and Medical
ITU	International Telecommunications Union

LAN	Local Area Network
LEO	Lower Earth Orbit
LOS	Line of Sight
LSB	Least Significant Bit
MSB	Most Significant Bit
MSK	Minimum Shift Keying
OBC	On Board Computer
OSI	Open Systems Interconnection
OTcl	Object-Oriented Tool Command Language
PNG	Portable Network Graphics
PPP	Point-to-Point Protocol
PRN	Pseudo Random Noise
PSK	Phase Shift Keying
QPSK	Quadrature Phase Shift Keying
RF	Radio Frequency
RRP	Round Robin Polling
RS	Reed-Solomon
RTS	Request to Send
SNR	Signal-to-Noise Ratio
SSID	Secondary Station Identifier
SVG	Scalable Vector Graphics
TDMA	Time Division Multiple Access
TNC	Terminal Node Controller
UHF	Ultra High Frequency
VHF	Very High Frequency

Scientific Constants

c	Speed of Light (299793.07 $\frac{m}{s}$)
k	Boltzmann Constant (1.3806488 $\times 10^{-23}$ $kg\ m^2\ s^{-2}\ K^{-1}$)

Chapter 1

Introduction

1.1 Background

The design of a nano-satellite's communication systems often determines a mission's scope. Limited data throughput from a satellite mission can be an issue for missions that generate large quantities of data. Nano-satellites are mostly launched at Lower Earth Orbit (LEO) altitudes which vary from 300 km to 2000 km. The low altitude results in short orbit periods, which lead to a limited communication period between a satellite and ground station during a satellite pass. The available time for communication per orbit should therefore be used as efficiently as possible.

The Electronic Systems Laboratory (ESL) at the University of Stellenbosch proposed a concept named OuterNet [1], which is a satellite data relay constellation as in Figure 1.1. The OuterNet consists of 14 satellites evenly spaced in a 900 km circular equatorial orbit. The purpose of the network is to provide more frequent access times, which can vastly improve data throughput for a satellite mission. OuterNet is one potential solution to increasing satellite data throughput.

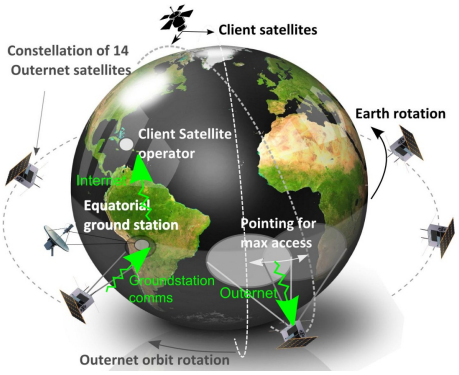


Figure 1.1 – The OuterNet system [1].

Nano-satellite data transmissions can be categorised into two fields, namely payload data and telecommands. Payload data consists of all the scientific information generated by the satellite and can include images and sensor information. Payload data is often only transmitted at set intervals, as the typically large volumes of data swamp the communication resources during this time. Telecommands are sent to satellites to operate and control the satellite and satellite sub-systems. These commands are usually of a small size which is quick to transmit, but it requires fast and reliable reception. The choice of network communication protocol influences the reliability and throughput of the communication link to a large extent.

There is limited information available regarding the performance of network protocols employed in satellite missions. This is a crucial area of study, especially when the feasibility and improvements of a systems such as OuterNet are evaluated. AX.25 [10] is one of the most commonly used nano-satellite communication protocols [14]. It has its roots in the amateur radio community. It is a simple and easy-to-use protocol, especially when operated in the connectionless mode. Version 2.0 of the AX.25 protocol definition was released in 1984 with the most recent version (2.2) being published in 1998.

Even though AX.25 is such an extensively used protocol, there is almost no network simulation data available. It is difficult to compare the performance of AX.25 with other protocols due to the limited available network simulation information. Comparing available satellite network protocols could assist with optimising future satellite communication reliability and throughput characteristics, especially in the nano-satellite field.

1.2 CubeSat

The CubeSat project began in 1999 as a collaborative effort between the California Polytechnic State University and Stanford University's Space Systems Development Laboratory. The purpose of the project was to provide a standard design for nano-satellites that would reduce cost and time, increase accessibility to space, and sustain frequent launches [15]. This concept has been a massive success. It provides a platform for regular new developments and has made satellite design and launching much more accessible to institutions such as universities, high schools and private firms.

A CubeSat is a 10 cm cube with a maximum weight of 1.33 kg [15]. This specification facilitates the development of standard launch vehicles that are purpose-built for CubeSats. Launch safety has also improved since all CubeSats will have similar shapes and weights, which translate to standard launching requirements. A partial CubeSat stack can be viewed in Figure 1.2.

The CubeSat movement has spawned a CubeSat industry consisting of various hardware developers and manufacturers. Some of these developers originated from academic institutions where CubeSats were researched and developed.

CubeSat projects typically make use of Commercial Off The Shelf (COTS) components rather than building custom satellite components, as it reduces cost and development time. However, custom hardware is often built for research or experimental payloads. This stim-

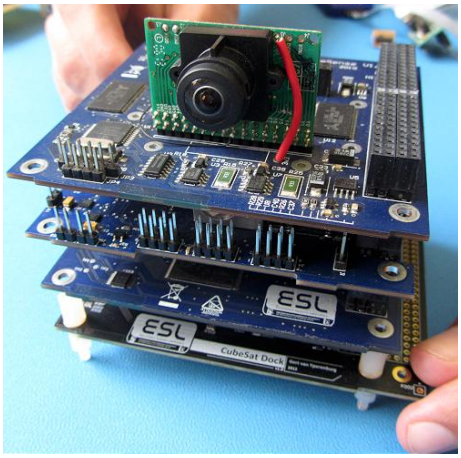


Figure 1.2 – A partial CubeSat hardware stack.

ulates the creation of innovative commercial CubeSat sub-systems. With the rapid development of the CubeSat environment, companies have started to produce purpose-built CubeSat hardware which can be acquired as a single functional unit. Entire 10×10 cm units can be purchased in the form of radio transceivers, Attitude Determination and Control System (ADCS) units, On-Board Computers (OBC) and power supply units, amongst others. The procurement of CubeSat standardised sub-systems enables resources to focus on the research related payloads.

1.3 Previous Work

Some research has been done on satellite network performance and modelling [16][17], but nearly all of this research was aimed at high level protocol optimisations. Plenty of satellite research is focused on the use of TCP/IP in a space environment [18][19][20]. Tools have been developed for protocol analysis. These tools are for high level protocols and are highly application-specific, for example the OMNeT++ based simulation tool for DVB-RCS and DVB-S2 [21].

Previous network simulation research, conducted at Stellenbosch University, includes an investigation into the optimisation of low-speed communication protocols [22]. This work, which focused on terrestrial networks, was implemented in DESMO-J, which is a Java based discrete-event simulation framework. Recent satellite communication research focused on selecting an optimal communication strategy for LEO satellites [8]. The aforementioned work is similar to this project, but it was aimed towards the selection of modulation schemes and channel access methods, whereas this thesis aims to optimise network throughput on a protocol level.

Limited research has been done on low-level satellite communication optimisation specifically. The research mostly focused on optimising TCP/IP layers in satellite links [23][24][25]. Other research studies address very specific problems, such as VoIP over satellite links [26].

Most protocol optimisation papers focus on very specific protocols, thus limiting the application of the optimisations. Single link analysis is also rarely investigated. A wider and more abstract protocol optimisation research approach is usually followed.

There is little to no research on the performance of the commonly used CubeSat (nanosatellite) network protocols. The AX.25 protocol is extensively used within the CubeSat community, but no simulation results could be found for this protocol.

1.4 Past CubeSat Missions

It is important to understand the common approaches towards certain CubeSat design aspects, specifically the design of the communication system. Klofas *et al.* [14] did a survey to document the communication system designs of past CubeSat missions specifically. This survey contains extensive detail on the communication designs of various CubeSat missions. For example, until July 2014, a total of 164 CubeSat missions have been documented [12]. Klofas maintains a summary of all CubeSat missions and updates this summary on a regular basis. The data of the past CubeSat missions can be viewed in Appendix A.

The survey data shows certain trends in the CubeSat environment, for example the preferred CubeSat parameters for the following: satellite size, communication transmission power rating, transmission baud rate, modulation technique, antenna type and communication frequency. Some of these trends can be attributed to good practice and others to technological limitations. Some of the trends found in the Klofas survey are discussed in the following section.

Table 1.1 – CubeSat sizes. Adapted from Ref. [12].

CubeSat Size	Percentage of CubeSat Missions
1U	58.53%
1.5U	3.05%
2U	6.71%
3U	25.61%
3U+	4.88%

CubeSat Sizes: As can be seen from Table 1.1, most of the past CubeSat missions were 1U in size . It is the most affordable and achievable satellite for most institutions. Smaller radios and antennas with lower power requirements are required for 1U missions. However, the 1U satellites' size and weight constraints force developers to invent intelligent solutions.

Table 1.2 – CubeSat communication power (not all missions disclosed transmission power). Adapted from Ref. [12].

Transmission Power (mW)	Percentage of CubeSat Missions
100	1.83 %
200	5.49 %
300	3.05 %
400	3.05 %
500	9.15 %
600	1.22 %
700	0.00 %
800	2.44 %
900	0.06 %
1000	37.72 %
1600	2.44 %
2000	4.27 %
3000	1.83 %

CubeSat Transmission Power: Transmission power is an important factor in system and communication design. A low transmission power requirement reduces overall system power requirements, but communication reliability could be compromised. The objectives of the specific CubeSat mission therefore determine the transmission power requirements. Table 1.2 indicates that most CubeSats are designed to use 1 W of transmission power or lower. The larger transmission powers are more often used on 3U satellites, as they have more room for batteries and surface area for solar panels.

Table 1.3 – Most common CubeSat baud rates. Adapted from Ref. [12].

Transmission Baud	Percentage of CubeSat Missions
1200	31.71 %
2400	1.83 %
4800	2.44 %
9600	22.56 %

CubeSat Baud Rates: Lower transmission frequencies produce better signal propagation characteristics, compared to higher transmission frequencies. However, the disadvantage of using lower transmission frequencies is the reduced bandwidth that is available and which results in lower data transmission rates. The requirements for transmission power, data rates and frequencies depend highly on the objective of a specific CubeSat mission. Table 1.3 shows that CubeSats tend to use lower baud rates, such as 1200 kbps. The missions

that make use of higher transmission frequencies (2.4 GHz+), achieve transmission rates in the upper kbps to Mbps range, but this is not achievable with the average CubeSat mission. The 400 MHz band is available for amateur radio broadcasting, within certain transmission power limits. The advantage of lower transmission frequencies is better signal propagation characteristics than higher transmission frequencies. However, the disadvantage is the reduced frequency bandwidth available which leads to lower data transmission rates. The requirements for transmission power, data rates and frequencies depend highly on the objective of a specific CubeSat mission.

Table 1.4 – Most common CubeSat modulation techniques. Adapted from Ref. [12].

Modulation Technique	Percentage of CubeSat Missions
GMSK	6.10 %
MSK	4.88 %
AFSK	19.51 %
FSK	26.83 %
GFSK	4.27 %

CubeSat Modulation Techniques: The most popular modulation techniques for CubeSats are variations of FSK modulation as shown in Table 1.4. The radios that use these modulation techniques are commercially available as integrated CubeSat units. FSK signals are relatively easy to modulate and demodulate and provide adequate performance for satellite communication channels.

Table 1.5 – Most common CubeSat antennas. Adapted from Ref. [12].

Antenna Type	Percentage of CubeSat Missions
Dipole	34.75 %
Monopole	26.83 %
Patch	18.29 %
Turnstile	12.80 %

CubeSat Antennas: Various types of antennas are used for CubeSat missions, as can be seen in Table 1.5. The most effective designs are dipole and monopole antennas, which serves on the majority of the CubeSat missions. Higher transmission frequencies are desirable as higher frequencies lead to smaller antenna sizes such as patch antennas. Patch antennas are efficient and take up little space compared to other antenna options - which is a major advantage - especially on a small 1U CubeSat.

CubeSat Transmission Frequencies: The world's Radio Frequency (RF) spectrum is divided into multiple bands, including the amateur bands and the Industrial, Scientific and Medical (ISM) bands. The available spectrum can be viewed in Figure 1.3.

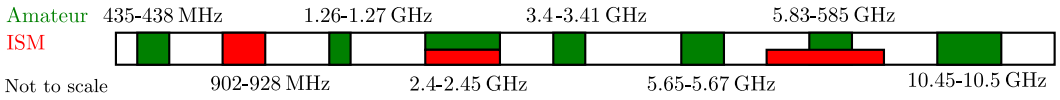


Figure 1.3 – Radio frequency spectrum division. Green sections are amateur radio bands and red sections are ISM bands. Adapted from Ref. [2].

Most CubeSats use the 435 MHz to 438 MHz amateur band. The use of an amateur band holds many advantages. There is no licensing costs compared to other bands. There is also a large number of radios available for amateur radio applications. The data transfer rates and power requirements are also appropriate for CubeSat missions when making use of lower frequency amateur bands. Lastly, amateur bands has the optional benefit of using the amateur radio community to gather data from satellites. Some larger CubeSat missions made use of the 2.4 GHz band in order to achieve a higher data throughput rate. These observations are confirmed by the results in Table 1.6.

Table 1.6 – Most common CubeSat communication frequencies. Adapted from Ref. [12].

Frequency Range	Percentage of CubeSat Missions
100+ MHz	10.98 %
400+ MHz	45.12 %
900+ MHz	4.27 %
2+ GHz	10.37 %

CubeSat Network Protocols: Nearly 55% of all the past CubeSat missions listed in Ref. [12] made use of the AX.25 network communication protocol. The other missions either used proprietary protocols, experimented with IP based protocols or did not disclose any information regarding the implemented network protocol. AX.25 is most certainly a standard in the industry.

The strategies for optimising the CubeSat communication design are limited by the strict CubeSat hardware specifications. This leaves limited options to improve the communication strategies of CubeSats due to the hardware limitations. Investigating potential protocol improvements will enable the efficient use of the limited communication hardware.

1.5 Objective and Contribution

The initial objective of this thesis was to research and design an optimal network protocol strategy for nano-satellite communications. Upon the investigation of available network simulation tools (see Section 2), it became evident that the development of an adequate simulation tool was required. The objective of this thesis then shifted to the development of a modular and easy to use satellite network simulation tool and an investigation into an optimal network protocol strategy for nano-satellite communications.

Both of the mentioned outcomes were achieved. A network simulation tool, named SatSim, was successfully developed in Python. SatSim was verified by duplicating results achieved with other simulators and comparing SatSim results with theoretical expectations. SatSim was then used to conceive an optimal communication protocol strategy for nano-satellites. The AX.25 and FX.25 protocols were implemented and simulated with documented results. An optimal solution was then derived from a combination of the mentioned protocols.

1.6 Thesis Structure

The chapters in this thesis is structured as follows:

Chapter 1: General overview and motivation for research.

Chapter 2: A brief overview of the available network simulation tools and the rationale for developing a new simulation tool.

Chapter 3: Documents the development process and internal operations of SatSim.

Chapter 4: An investigation into available network protocols for nano-satellites which includes simulations of AX.25 and FX.25.

Chapter 5: An overview and design of an optimal communications protocol which is a hybrid protocol consisting of AX.25 and FX.25.

Chapter 6: The conclusions, recommendations and future work is summarised in the final chapter.

Chapter 2

Simulation Tools

The defining characteristic of a satellite network is its movement around the earth and the wireless nature of the communication channel. Thus, there are a lot of dynamics involved when simulating a satellite network. The simulation of the communication network, as well as the satellite dynamics need to be carefully planned.

2.1 Introduction

There are a wide variety of simulation tools available, ranging from comprehensive network simulators (for example ns2, ns3, OPNET) to programming frameworks for discrete-event simulation (for example SimPy, OMNeT++, DESMO-J). A network simulator is software that is purpose-built for simulating networks. A network simulator often contains an arrangement of pre-developed packages that can be used to simulate a variety of topologies and protocols. A programming framework for discrete-event simulation is not a pure network simulator. It provides only a framework that can be applied to any process to simulate it in discrete time.

2.2 Network Simulators

2.2.1 ns2

'ns2' [27] is an open source network simulation platform. It provides a wide variety of network protocols, queuing methods and node types, including LEO and Geostationary Earth Orbit (GEO) satellite models. The satellite models are not entirely accurate, for example the error due to GEO satellites' drift from their designated positions, due to gravitational perturbations is not modelled. This will rarely be an issue, unless it is the key point of study.

The simulator provides all the basic tools required to create a satellite network simulation. Satellite constellations, Inter Satellite Links (ISL), ground-to-satellite links and elevation masks (smallest communication angle) can be defined. Communication link hand-off is natively handled in ns2 and parameters such as hand-off times can be defined. Routing

uses a centralised routing agent. After each topology change, a centralised routing agent determines the global network topology, computes new routes for all nodes and uses the routes to build a forwarding table on each node. This kind of abstract routing can lead to causality violations which is counter-intuitive for realistic results.

Another drawback of ns2 is the effort required to implement new models. The simulator consists of an Object-Oriented Tool Command Language (OTcl) scripting front-end, wherein the simulation is configured, while the simulation and modules are all implemented in C++. Very specific C++ file structures are required to implement new modules, and various header files require modification to add new modules successfully. The ns2 simulator requires re-compilation after any changes to source files are made. A good understanding of the ns2 simulator's structures is required to operate it successfully.

2.2.2 ns3

'ns3' [28] is the replacement for ns2 and is under continued development. The focus of ns3 is largely on internet applications. The ns3 simulator currently lacks the satellite modules that are available in ns2. It should be possible to port the modules manually, but this would require a considerable amount of work and testing. The software developer needs to understand ns3's structures well, in order fully understand and predict the simulator's user-friendliness.

2.2.3 OPNET Modeller IT Guru Academic Edition

'OPNET Modeller' is a commercially available network simulator software package [29]. The OPNET IT Guru Academic Edition is a free version for academic institutions on a renewable license agreement. The cost of the full commercial version is excessive for a project of this scale.

The Academic Edition provides pre-built models of protocols and devices. It allows the user to simulate different network topologies. The set of protocols and devices are fixed and a user cannot create new protocols nor modify the behaviour of existing ones, although parameters can be modified. Some variables include Request To Send (RTS) thresholds, packet arrival rates and data rates. The Academic Edition is limited to simulating 50 million events. Events are actions in the network, for example receiving a packet or the occurrence of a timeout. A wireless Local Area Network (LAN) network with 10 devices all generating a high load will reach 50 million events in about 5 minutes of simulation time. It is important to note that OPNET has recently been acquired by River Bed Technologies and the focus area of the software may change. The limitations and uncertain future of the academic package makes OPNET an inadequate solution, unless a fully licensed version can be acquired.

2.3 Discrete Simulation Frameworks

2.3.1 OMNeT++

OMNeT++ is a discrete-event simulation environment based on the C++ programming language. Its primary application area is the simulation of communication networks, but due to its generic and flexible architecture, it has been used successfully in other areas, for example the simulation of complex IT systems, queuing networks and hardware architectures. OMNeT++ provides a component architecture for models. Components (modules) are programmed in C++, then assembled into larger systems and models using a high-level language (NED), in a similar approach to ns2. Although OMNeT++ is not a network simulator in itself, it is currently gaining widespread popularity as a network simulation platform in the scientific community. Research or testing conducted in OMNeT++ can take some time because it is not a dedicated network simulator. The complexities involved in modelling networks could house some difficulty for less experienced developers.

2.3.2 SimPy

SimPy (Simulation in Python) is an object-oriented, process-based, discrete-event simulation framework for Python. SimPy provides a framework to build a discrete-event simulation. Events are created and placed in an event list which is ordered consecutively. The events are then executed sequentially and new events can be dynamically added as the simulation progresses. Additional functionality is provided by means of Python modules, for example random variables are provided by the standard Python random module.

Python has a large community with many active developers that provide packages that can assist with scientific methods, graphical interfaces and visualisation of data. Python is a high-level language (similar to Java and C++) and is trademarked by its simplicity and ease of use. Python does not have a very involved compiling process, as the scripts are compiled into byte codes. This allows for quick modifications to scripts, but at the cost of some execution performance. This is a potential issue if simulations take too long to execute, but there are techniques available to address this problem.

2.3.3 Java-Based Network Simulators

The Java programming language has numerous simulation framework packages available including MASON, DESMO-J and SimJava (amongst others). The large number of discrete-event simulators indicate that there is no clear preferred package and some experimentation will be required to find the most suitable simulation package. However, past research at Stellenbosch University [22] achieved good results with DESMO-J. It would potentially be a better option to make use of Python than Java, as there will be better support available for a Python based environment. Python also includes many standard and community packages that can assist with the development of a simulator.

2.4 Conclusion

After experimenting with some of the aforementioned simulators, SimPy and Python were chosen as the software development platforms for this thesis. Some of the deciding factors were:

- SimPy has a reputation for being a reliable and easy-to-use, discrete-event simulation package.
- The Python language is easy to learn and implement.
- Both Python and SimPy have well documented support files, tutorials and supportive development.
- Python has additional packages that will assist greatly with the development of a simulator.

Even though the development of a simulation package in Python may be more complicated than simply using a dedicated simulator software package, the benefits of increased development flexibility, in terms of what can be measured and tested, makes it worth the effort.

Chapter 3

Simulator Overview and Design

3.1 Overview

An open-source simulator, SatSim, was developed for this project. It is a modular, discrete-event network simulator, developed in Python. The simulator can visualise a simulation as shown in Figure 3.1. The visualisation provides text based feedback, as well as graphical feedback to the user. This assists greatly with the verification of a simulation configuration.

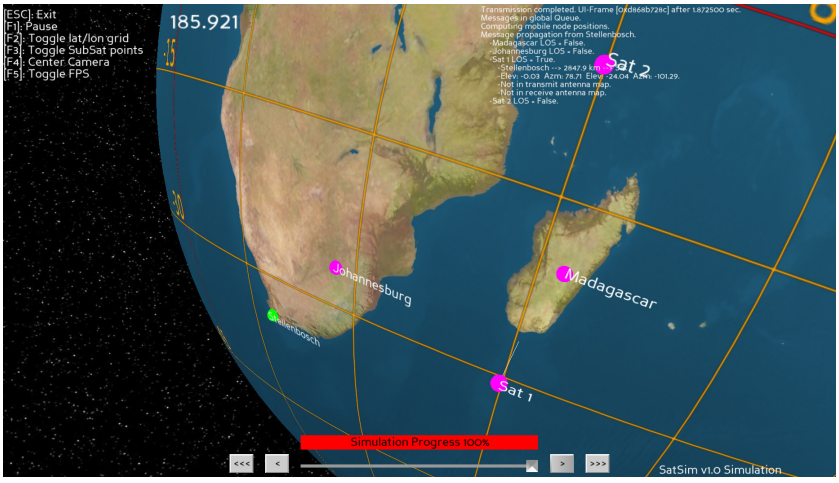


Figure 3.1 – SatSim visualisation.

Larger simulations are run in scripts without visualisation or feedback regarding individual events. This is only done after an initial configuration has been fully tested. Multiple simulation runs are required to test different parameters. The full source code and documentation for SatSim are available on GitHub (https://github.com/jhleroux/SatSim).

3.1.1 Additional Python Packages in SatSim

One of the advantages of using Python is the large and active development community. There are constant developments on new and existing packages. The packages vary from graphical - to scientific applications. The existing packages assisted the development of SatSim, as it was unnecessary to build all the elements of the SatSim package from scratch. Some of the community developed Python packages, that were used in the development of SatSim, are described next.

3.1.1.1 PyEphem

PyEphem [30] is a Python package that provides astronomy computations. The software algorithms are the same as in the XEphem software package [31]. The XEphem code, which is written in C, is wrapped in Python to produce PyEphem. PyEphem allows the user to create satellite objects and propagate them through time. The earth satellite models are based on the NORAD SGP4/SDP4 orbital models. Regular PyEphem updates address user reported errors or suggestions.

PyEphem provides additional functionality that is not required for this simulation, such as the ability to model other celestial bodies as well. Another package named Python-SGP4 exists which provides SGP4 modelling, but it is also possible to implement the SGP4 modelling by hand. These work around solutions can reduce some of the limitations of the PyEphem package and can form part of future work and developments.

3.1.1.2 SimPy

SimPy [32] has been discussed in Section 2.3.2. Version 2.3 of SimPy was used at the start of the project and was upgraded to version 3.0.4 during late 2013. This upgrade involved the reprogramming and restructuring of most of the SimPy code in the simulation. The upgrade was implemented, as it vastly improved the readability of the code and the structure of the simulation.

3.1.1.3 GUI and Visualisation

Initial visualisations were done in a sub-package of Matplotlib named Basemap Toolkit [33]. This package can render 2D or isometric renderings of the earth. This provided a visual way of verifying if initial geometrical implementations functioned correctly. However, it became clear that a real time 3D environment was required for proper visualisation of satellite orbits and motion.

Multiple packages were experimented with to find an elegant 3D solution to rendering a planet with orbiting satellites. Amongst all the packages tested, MayaVi was the most impressive, but it was a scientific plotting package and held some severe rendering limitations for future expansions. Panda3D [34] eventually proved to be the desired solution as it features a full 3D engine with a Python interface. This removed many limitations and allowed for a 3D visualisation and a functional GUI within the same software package.

The SatSim GUI was originally rendered in Kivy, which is a cross-platform user-interface framework. This package provided a visually impressive GUI, but development time was excessive due to Kivy's lack of a visual design tool. The GUI was then shifted to Qt4 which provided a fast, easy to modify and lightweight solution, with a visual design tool. This not only sped up development, but the GUI system structure eased any future additions and modifications tremendously.

3.1.2 Simulator Structure

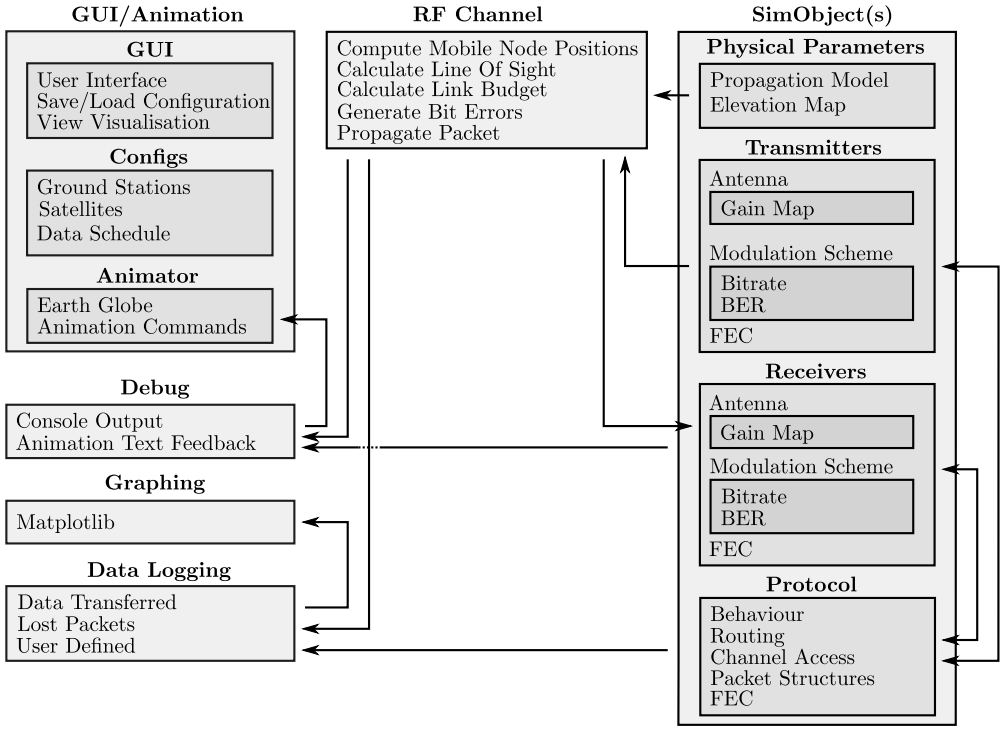


Figure 3.2 – SatSim structure.

A high-level overview of the SatSim simulation structure can be viewed in Figure 3.2. A simulation is structured around a main simulation script, which calls in all the necessary modules to perform a simulation. These scripts determine the behaviour of the simulation. The main functional elements of a script are the simulation objects, in the form of satellites and ground stations, the RF channel and the data schedule. Scripts provide plenty of flexibility to the user. The user can run multiple simulations concurrently, plot data (framework for Matplotlib provided) and iterate through varying configurations in a single script. SatSim also provides an optional GUI which allows for fast prototyping and testing of certain configurations.

3.1.3 Frame Structure

There are two options to represent frames in SatSim. Firstly, a frame can be represented as a binary string of ones and zeros. This allows for the decoding of bit streams within the simulation, which is required for realistic simulations. Each frame in this format is stored as an error-free frame and an erroneous frame. The erroneous frame is generated once a frame passes through the communication channel. Both the error-free and the erroneous frames are required when Forward Error Correction (FEC) schemes are implemented, where the error-free frame is required as reference.

The alternative option is to represent frames as combinations of fields. This representation limits the possibilities of bit-level operations and the decoding of bit streams. Individual bit error positions are not calculated and only provided as a quantity per field in a frame. This is a more abstract method and its use is limited to the desired area of investigation and complexity of the protocol.

3.1.4 Object-Oriented Design

SatSim uses an object-oriented design approach. An object-oriented design allows for an easy-to-use and modular programming environment when it comes to large software packages, especially when compared to other programming paradigms such as imperative programming. The use of object-oriented programming languages by other simulation tools are a good indicator of the popularity of object-oriented design with simulation tools.

Object-oriented design simplifies logical analysis of a simulation, as each entity in the simulation – be it the communication channel, or a satellite – can be tested individually. This approach allows the user to modify or replace an object, as long as the interface stays the same. A good example would be if the user requires a specific type of transmitter: the standard transmitter class can be modified to fit the user's needs.

3.1.5 Multi-Processing

Most modern computer processors contain more than one processing core. This is a valuable characteristic for simulations. SatSim has been designed to run a single simulation in a single thread within a single process. This makes it possible to run multiple simulations concurrently on a single processor. Executing multiple simulations concurrently allows the user to build up a statistical average or test multiple configurations in less time than running each simulation in sequence.

The single process design choice was chosen, since there is limited room for multiprocessing within a single simulation, as discrete-events occur sequentially, which prohibits processing of events in parallel. Certain computations such as the calculation of geometry (distances, altitudes etc.) can be processed in parallel, but this will only start yielding improved performance if a large number of objects are used in the simulation, as the computational overhead involved with multiprocessing can be large.

It is also possible to execute more simulation processes than the number of processing cores on a processor. For example, it is possible to execute 100 simulations in parallel on a 4-core processor, however, memory usage may become an issue, depending on the size of each simulation. Operating system overhead can reduce simulation execution performance, when trying to manage such a large number of processes simultaneously.

An optimal approach to implementing multiprocessing is to create a worker process for each processing core. Each worker process will then execute one simulation after the other, until all the configurations have been simulated.

3.1.6 Documentation

An aspect that is often overlooked in software development, is the software documentation. SatSim is thoroughly documented and makes use of Sphinx [35] to generate HTML based documentation. This allows the user to easily navigate and access relevant documentation when required.

3.2 Simulation Geometry

Two unique features of a satellite network are the movement of the satellites and the relative angles and distances between communicating nodes. The elevation, azimuth and distance between two communicating objects are required to compile a detailed link budget for communication.

3.2.1 Satellite Orbiting Model and Ground Stations

PyEphem provides the satellite orbit model, as explained in Section 3.1.1.1. PyEphem provides objects for modelling observers (ground stations) and satellites orbiting the earth. Objects are also provided for planets and other celestial bodies, but it is not relevant to this project. Currently SatSim only supports circular orbits, even though it is possible to define elliptical orbits. The accuracy of elliptical orbits are not fully tested and elliptical orbits were not required for this project.

The PyEphem classes are wrapped inside SatSim classes to facilitate operation. This allows for a consistent interface for propagation models. The underlying code routines can easily be replaced if required.

3.2.2 Elevation, Azimuth and Distance

This section describes the calculations required for the elevation, azimuth and distance between two simulation objects. The mathematical derivations in this section are based on the space mission geometry calculations from Ref. [3]. The geometry calculations are required to determine if a signal is not blocked by any physical obstructions and to compile a detailed link budget to determine signal strength.

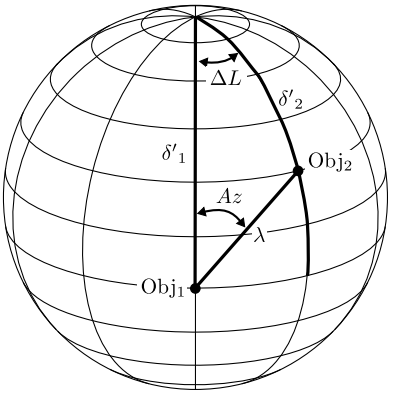


Figure 3.3 – Azimuth and λ calculation. Adapted from Ref. [3].

The geometry for these calculations are based on a spherical coordinate system with the origin located at the centre of the earth as is illustrated in Figure 3.3. Obj_1 and Obj_2 are the two points that are used for the distance and azimuth calculations. Obj_1 has a latitude of

δ_1, and Obj_2 has has a latitude of δ_2. Lastly, Obj_1 will always be chosen as the object with the lowest altitude, for example Obj_1 will be a ground station and Obj_2 will be a satellite. This is done to avoid complexity with the calculations as the equations require adjustments if Obj_2 is lower than Obj_1 due to the shift of the relative angles between the objects.

To determine the relative angles between the two objects, it is required to solve the variables in Figure 3.3, starting with the relative longitude angle, ΔL:

$$\Delta L = |\text{Longitude}_1 - \text{Longitude}_2| \tag{3.2.1}$$

The angle ΔL will always be the smallest angle between the two objects, which means that ΔL will always be smaller or equal to 180°. By applying the law of cosines for sides to the configuration in Figure 3.3, the arc angle between the two objects (λ) can be calculated:

$$\cos(\lambda) = \cos(\delta_1')\cos(\delta_2') + \sin(\delta_1')\sin(\delta_2')\cos(\Delta L) \tag{3.2.2}$$

But,

$$\delta_1' = 90° - \delta_1 \tag{3.2.3}$$

$$\delta_2' = 90° - \delta_2 \tag{3.2.4}$$

By substituting 3.2.3 and 3.2.4 into 3.2.2 the following can be calculated:

$$\begin{aligned} \cos(\lambda) &= \sin(\delta_1)\sin(\delta_2) + \cos(\delta_1)\cos(\delta_2)\cos(\Delta L) \\ \lambda &= \arccos\left(\sin(\delta_1)\sin(\delta_2) + \cos(\delta_1)\cos(\delta_2)\cos(\Delta L)\right) \end{aligned} \tag{3.2.5}$$

Equation 3.2.5 provides a solution for calculating λ (arc angle between the two objects). The azimuth angle from the north (Az) is also calculated by applying the law of cosines for sides and substituting in 3.2.3 and 3.2.4:

$$\begin{aligned} \cos(\delta_2') &= \cos(\lambda)\cos(\delta_1') + \sin(\lambda)\sin(\delta_1')\cos(\text{Az}) \\ \sin(\delta_2) &= \cos(\lambda)\sin(\delta_1) + \sin(\lambda)\cos(\delta_1)\cos(\text{Az}) \\ \cos(\text{Az}) &= \frac{\sin(\delta_2) - \cos(\lambda)\sin(\delta_1)}{\sin(\lambda)\cos(\delta_1)} \\ \text{Az} &= \arccos\left(\frac{\sin(\delta_2) - \cos(\lambda)\sin(\delta_1)}{\sin(\lambda)\cos(\delta_1)}\right) \end{aligned} \tag{3.2.6}$$

Equation 3.2.6 calculates the azimuth angle from the north from Obj_1 to Obj_2. The azimuth has to undergo the following modifications to ensure the correct value depending on the relative position between the two objects.

$$\text{Azimuth } (\text{Obj}_1 \text{ to } \text{Obj}_2) = \begin{cases} \text{Az} & \text{if } (\text{Longitude}_1 - \text{Longitude}_2) < 0 \\ -\text{Az} & \text{if } (\text{Longitude}_1 - \text{Longitude}_2) \geq 0 \end{cases}$$

$$\text{Azimuth } (\text{Obj}_2 \text{ to } \text{Obj}_1) = \begin{cases} \text{Az} - 180° & \text{if } (\text{Longitude}_1 - \text{Longitude}_2) < 0 \\ \text{Az} + 180° & \text{if } (\text{Longitude}_1 - \text{Longitude}_2) \geq 0 \end{cases}$$

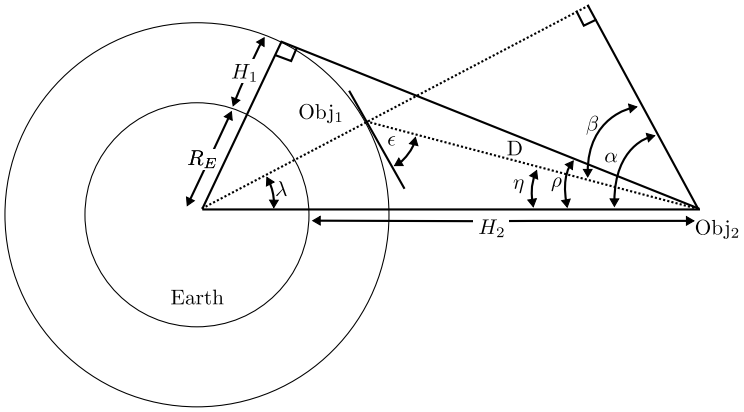


Figure 3.4 – Calculations for η and ρ. Adapted from Ref. [3].

Now it is required to calculate the variables (ρ, α, η, β, ϵ and D) in Figure 3.4 to determine the elevation:

$$\sin(\rho) = \frac{R_E + H_1}{R_E + H_2}$$
$$\rho = \arcsin\left(\frac{R_E + H_1}{R_E + H_2}\right) \tag{3.2.7}$$

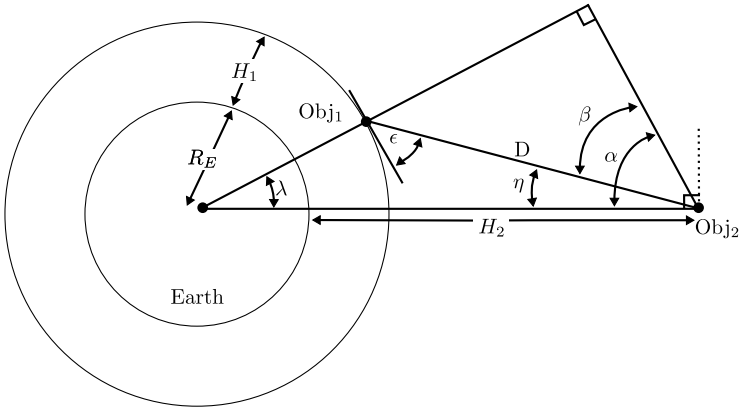


Figure 3.5 – First case for calculation of ϵ and D where $\lambda <= 90°$. Adapted from Ref. [3].

Where $\lambda <= 90°$ as in Figure 3.5:

$$\alpha = 180° - 90° - \lambda$$
$$\alpha = 90° - \lambda \tag{3.2.8}$$

and,

$$\tan(\eta) = \frac{\sin(\rho)\sin(\lambda)}{1 - \sin(\rho)\cos(\lambda)}$$
$$\eta = \arctan\left(\frac{\sin(\rho)\sin(\lambda)}{1 - \sin(\rho)\cos(\lambda)}\right) \tag{3.2.9}$$

$$\beta = \alpha - \eta \tag{3.2.10}$$

$$D = \frac{(R_E + H_2)\sin(\lambda)}{\sin(\eta)} \tag{3.2.11}$$

The elevation between the two objects are:

$$\text{Elevation } (\text{Obj}_1 \text{ to } \text{Obj}_2) = \epsilon = \beta \tag{3.2.12}$$

$$\text{Elevation } (\text{Obj}_2 \text{ to } \text{Obj}_1) = -(90^\circ - \eta) \tag{3.2.13}$$

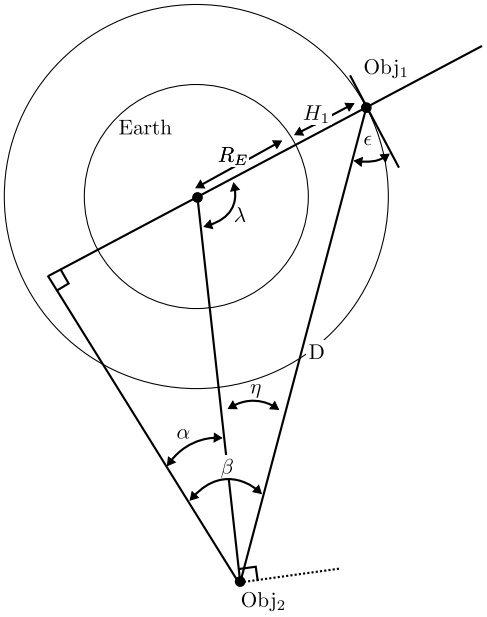


Figure 3.6 – Second case where $\lambda > 90^\circ$. Adapted from Ref. [3].

And when $\lambda > 90^\circ$, as illustrated in Figure 3.6, the calculation for η and D stay the same, but α, β and ϵ changes,

$$\alpha = 90^\circ - (180^\circ - \lambda)$$

$$\alpha = \lambda - 90^\circ \tag{3.2.14}$$

$$\beta = \alpha + \eta \tag{3.2.15}$$

$$\text{Elevation } (\text{Obj}_1 \text{ to } \text{Obj}_2) = \epsilon = -\beta \tag{3.2.16}$$

$$\text{Elevation } (\text{Obj}_2 \text{ to } \text{Obj}_1) = -(90^\circ - \eta) \tag{3.2.17}$$

3.2.3 Line Of Sight Calculations

Line of Sight (LOS) is required between two objects to successfully communicate through a RF link in SatSim. There are two potential obstructions to successful communication: local obstructions (terrain, buildings) and obstruction by the earth (when the ground station and the satellite LOS is obstructed by the earth). The former is relevant to ground stations and is unique to each ground station. Local obstructions will be modelled through an elevation map.

3.2.3.1 Elevation Map

The elevation map provides a way of specifying the visible communication range of a simulation object on the surface of the earth. An example of an elevation map can be viewed in Figure 3.7.

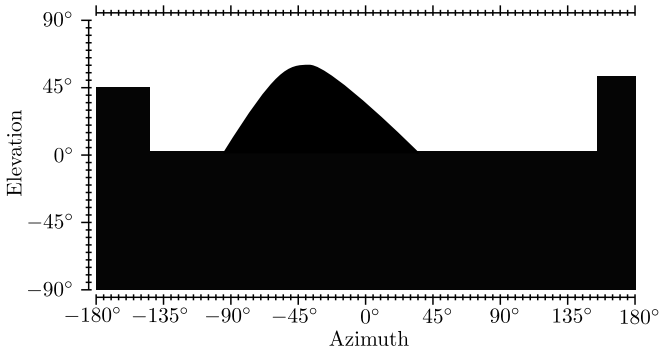


Figure 3.7 – Example elevation map.

The elevation map is allocated to a ground station and is referred to whenever communications are attempted. The satellite's elevation and azimuth are provided to the ground station object and is then compared to that specific point on the ground station's elevation map to determine if the view is obstructed or not. An elevation of 0° is parallel to the surface of the earth with 90° (away from earth) and −90° (towards earth) being perpendicular to the surface of the earth. As explained in Figure 3.3, the azimuth is measured from the north, where an azimuth angle of 0° points exactly north. The elevation map is stored as a 360 by 181 pixel Portable Network Graphics (PNG) image. This format provides a resolution of 1 degree in both azimuth and elevation values. The size of the PNG image can be adjusted to accommodate a higher resolution map if required. The Python image libraries fully support the PNG image format and image tools, such as Inkscape, can easily generate PNG files. A Scalable Vector Graphics (SVG) file is provided to assist with designing new elevation maps.

3.2.3.2 Earth Obstruction

A Cartesian coordinate system is used to calculate if there is a LOS between two simulation objects. The earth is modelled as a sphere, even though it is actually flatter at the poles. This inaccuracy in the model will have little effect on simulations and will be assumed as negligible. The simulation objects are modelled as points in space. The goal is to determine if the LOS between the two objects is obstructed by the earth.

A line in three-dimensional space is defined as follows:

$$x = X + et$$
$$y = Y + ft$$
$$z = Z + gt$$

with the line going through the point (X, Y, Z) with a direction vector (e, f, g) and t as any real value. If there are two points $p_1 = (x_1, y_1, z_1)$ and $p_2 = (x_2, y_2, z_2)$ in three-dimensional space, the line between these points can be defined as:

$$x = x_1 + (x_1 - x_2)t \tag{3.2.18}$$
$$y = y_1 + (y_1 - y_2)t \tag{3.2.19}$$
$$z = z_1 + (z_1 - z_2)t \tag{3.2.20}$$

A sphere (earth in the simulation) in three-dimensional space can be defined as follows:

$$x^2 + y^2 + z^2 = R^2 \tag{3.2.21}$$

with R the radius of the sphere. To determine if the line defined in Equations 3.2.18 - 3.2.20 intersects the sphere, the line equations are substituted into Equation 3.2.21 which yields:

$$(x_1 + (x_1 - x_2)t)^2 + (y_1 + (y_1 - y_2)t)^2 + (z_1 + (z_1 - z_2)t)^2 = R^2 \tag{3.2.22}$$

$$\begin{aligned} &x_1^2 + 2x_1^2 t - 2x_1x_2t + (x_1^2 - 2x_1x_2 + x_2^2)t^2 + \\ &y_1^2 + 2y_1^2 t - 2y_1y_2t + (y_1^2 - 2y_1y_2 + y_2^2)t^2 + \\ &z_1^2 + 2z_1^2 t - 2z_1z_2t + (z_1^2 - 2z_1z_2 + z_2^2)t^2 = R^2 \end{aligned} \tag{3.2.23}$$

$$\begin{aligned} &(x_1^2 - 2x_1x_2 + x_2^2 + y_1^2 - 2y_1y_2 + y_2^2 + z_1^2 - 2z_1z_2 + z_2^2)t^2 + \\ &(2x_1^2 - 2x_1x_2 + 2y_1^2 + 2y_1y_2 + 2y_1y_2 + 2z_1^2 - 2z_1z_2)t + \\ &(x_1^2 + y_1^2 + z_1^2 - R^2) = 0 \end{aligned} \tag{3.2.24}$$

Equation 3.2.24 can be solved as a standard quadratic equation with:

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{3.2.25}$$

with,

$$a = x_1^2 - 2x_1x_2 + x_2^2 + y_1^2 - 2y_1y_2 + y_2^2 + z_1^2 - 2z_1z_2 + z_2^2$$
$$b = 2x_1^2 - 2x_1x_2 + 2y_1^2 + 2y_1y_2 + 2y_1y_2 + 2z_1^2 - 2z_1z_2$$
$$c = x_1^2 + y_1^2 + z_1^2 - R^2$$

If Equation 3.2.25 has no solution, then there is no intersect between the line and the sphere. If it does have a solution, it will yield the values t_1 and t_2. These values can be substituted into Equation 3.2.18, 3.2.19 and 3.2.20 to provide the coordinates of the intersection points (i_1 and i_2) as follows:

$$i_1 = ([x_1 + (x_1 - x_2)t_1],\ [y_1 + (y_1 - y_2)t_1],\ [z_1 + (z_1 - z_2)t_1]) \tag{3.2.26}$$

$$i_2 = ([x_1 + (x_1 - x_2)t_2],\ [y_1 + (y_1 - y_2)t_2],\ [z_1 + (z_1 - z_2)t_2]) \tag{3.2.27}$$

The intersection points with the sphere have been calculated, but it is still uncertain if the intersection points are on the bounded line between points p_1 and p_2. To determine this, the following distances are calculated:

d_{00} = The distance between p_1 and p_2.

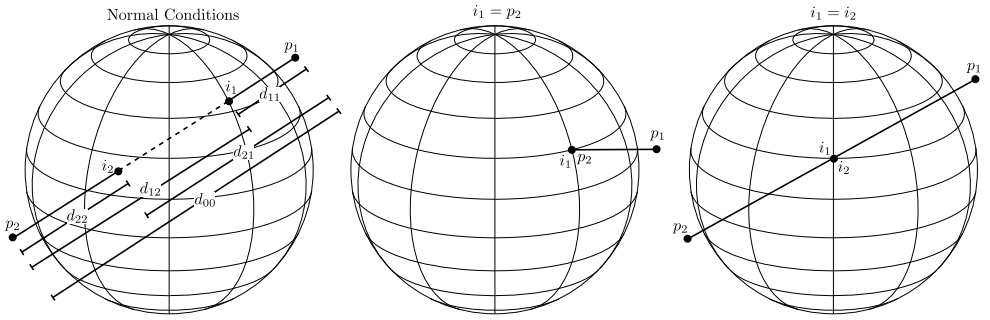
d_{11} = The distance between i_1 and p_1

d_{12} = The distance between i_1 and p_2

d_{21} = The distance between i_2 and p_1

d_{22} = The distance between i_2 and p_2

Figure 3.8(a) visually illustrates these distances.



(a) Line lengths. (b) Same point intersect. (c) Glancing intersect.

Figure 3.8 – Line of sight calculations.

The following rules determine if there is an point of intersection with the earth or not:

If $d_{11} + d_{12} <= d_{00}$, then i_1 is on the line.

If $d_{21} + d_{22} <= d_{00}$, then i_2 is on the line.

Two checks have to be done to finally determine LOS. The first check is to see if either point p_1 or p_2 is the same as i_1 or i_2. This occurs when one object is on the surface of the earth as in Figure 3.8(b). The final check is for when i_1 and i_2 are the same point, which means that the line glances off the surface of the earth, but does not intersect. This is illustrated in Figure 3.8(c), but it will have a minor effect on final simulation results as it can only happen under very specific circumstances.

3.2.4 Potential Future Improvements

Frequency Dependent Signal Propagation - A common principle applicable to any wave, in this case an electromagnetic wave, is the diffraction principle. Diffraction is in essence the bending of waves around obstructions such as a mountain or building. Diffraction can also be caused by the wave's movement through mediums of different densities. Diffraction often helps electromagnetic waves to propagate behind the "shadow" of large obstructions. This allows communications below the visible horizon and beyond the line of sight. The amount of diffraction is dependent on the frequency of the wave. The simulator currently relies only on line of sight communication and adding a diffraction model would allow for more accurate results.

Accurate Earth Model - The earth is currently modelled as a perfect sphere, while in reality it is flatter at the poles. The influence of a spherical earth model on network simulations is unknown and it would be worthwhile to investigate an implementation of a more realistic earth model.

3.3 Channel Properties, Modulation and Radio Antennas

This section explores the physical properties of radio communications that need to be modelled for the simulation. Some concepts that are introduced in this section are required in order to evaluate and calculate a dynamic link budget. The link budget calculation is discussed in Section 3.4.

3.3.1 Carrier Frequency and Bandwidth

In telecommunication engineering, the Friis transmission equation is used to calculate the power received by an antenna from another antenna, under ideal conditions. The Friis equation, in its simplest form, is written as follows [36]:

$$\frac{P_r}{P_t} = G_t G_r \left(\frac{\lambda}{4\pi R}\right)^2 \tag{3.3.1}$$

where, P_r the received power,
P_t the transmitted power,
G_t the transmitter gain,
G_r the receiver gain,
λ the wavelength of the carrier signal,
R the distance between the antennas.

From the Friis equation it can be deduced that the power of the received signal (P_r) will be severely influenced by the wavelength (λ) and the relative distance between the two antennas (R). The distance between the antennas will vary with the orbital path of the satellite. There is only a limited amount of control over the distance between antennas. The wavelength of the carrier signal can, however, be adjusted to specific requirements. The longer the wavelength, the larger the amount of received power will be. The signal frequency maintains the following relationship with the wavelength:

$$\lambda = \frac{c}{f} \tag{3.3.2}$$

where, λ the wavelength of the carrier signal,
c the speed of light,
f the carrier signal frequency.

By substituting the above relationship into equation 3.3.1 the following is obtained:

$$\frac{P_r}{P_t} = G_t G_r \left(\frac{c}{4\pi R f}\right)^2 \tag{3.3.3}$$

From this equation it can be seen that a signal with a lower frequency will result in a larger received power compared to a high frequency signal with the same transmitting power. A high frequency carrier signal requires more transmission power than a lower frequency signal in order to maintain the same signal strength at reception. This characteristic is also known as signal propagation loss. Even though a lower frequency signal experiences

less propagation loss, it is important to note that more bandwidth is available at higher frequencies. The influence of bandwidth on a communication channel can be examined by means of the Shannon-Hartley theorem. The Shannon-Hartley theorem defines the channel capacity (with noise) and is given below [37]:

$$C = B \log_2(1 + SNR) \tag{3.3.4}$$

where, C the channel capacity in bits per second (bps),
B the channel bandwidth in Hertz (Hz),
SNR the Signal-to-Noise ratio.

From Shannon-Hartley's theorem it can be seen that the channel capacity can potentially be doubled by doubling the signal bandwidth. This is only true if the SNR stays the same, as an increased bandwidth will introduce additional noise, which decrease the SNR. Section 3.4 will expand on the concept of SNR and how it is influenced by various factors.

3.3.2 RF Modulation Techniques

Signal modulation is used to encode digital (binary) data onto analogue signals. This technique is critical for wireless transmission, as the choice of modulation technique influences the chance of data corruption. There are many modulation techniques available with various advantages and disadvantages. Most of these techniques involve modifying a carrier signal to encode the digital data through variations in amplitude, frequency or phase.

All the theoretical Bit Error Rate (BER) equations in this section are referenced from the handbook compiled by the Joint Spectrum Center [38], which conveniently summarises the BER equations in a single document. These equations are also included in most communication text books and available via various online sources.

In the discussion below, there will be continuous references to the number of bits (k) and number of symbols (M). A 'symbol' is a certain combined state of frequency, amplitude and/or phase that represents a number of bits. If a symbol represents one bit, it is the binary case. The single bit (k=1) can either be a logical 1 or 0, thus rendering two symbol possibilities (M=2). If a symbol consists of two bits (k=2), there is a total number of four symbols (00, 01, 10 and 11), therefore M=4. The relationship between the number of symbols and bits is:

$$M = 2^k \tag{3.3.5}$$

For the case of k=3 (M=8), one signal state represents 3 bits which means that 3 bits can be transmitted with one signal state or symbol. This will give triple the bit rate than for k=1 (M=2). Nyquist's theorem gives the upper limit of the bit rate for a system defined by the following equation [39]:

$$\begin{aligned} C &= 2B \log_2(2^k) \\ C &= 2B \log_2(M) \end{aligned} \tag{3.3.6}$$

where C is the channel capacity in bps and B is the channel bandwidth in Hz. Since $\log_2(M)$ is the number of bits per baud, the maximum baud is then according to the Nyquist criterion:

$$R = 2B \tag{3.3.7}$$

where R is the baud rate (or symbol rate). By substituting this into Equation 3.3.6, the following is achieved:

$$C = R\log_2(M) \tag{3.3.8}$$

where C is the channel capacity in bps, R is the baud rate, and M is the number of symbols. This is a convenient way of determining the data rate for a selected baud rate and the number of symbols.

Various modulation schemes exist which transmits data as symbols. Some of the more commonly used schemes will be discussed next.

3.3.2.1 Phase Shift Keying

Phase Shift Keying (PSK) is a modulation technique where the phase of a carrier signal is varied to encode data. The number of symbols for the chosen modulation scheme will determine the number of phase values required to represent the data. The more phases required, the higher the possibility for data corruption as the phases are spaced closer together.

BPSK: Binary Phase Shift Keying (BPSK) is a form of PSK modulation where k=1 and M=2, which means the modulation consists of one bit representing two possible symbols. A basic illustration of BPSK can be viewed in Figure 3.9 where a phase change can be noted between each binary data value change.

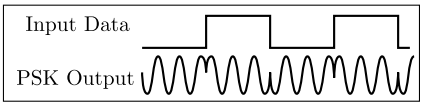


Figure 3.9 – Illustration of BPSK modulation.

The BER for BPSK modulation is as follows:

$$BER = \frac{1}{2}\,\text{erfc}\left(\sqrt{\frac{E_b}{N_o}}\right) \tag{3.3.9}$$

where 'erfc' is the complementary error function and $\frac{E_b}{N_o}$ is the energy-per-bit to noise-power-density-ratio. $\frac{E_b}{N_o}$ gives an indication of how much stronger the signal is compared to the background noise. $\frac{E_b}{N_o}$ relates back to the SNR as a ratio of the bandwidth and raw data rate.

A plot of the BER for BPSK can be viewed in Figure 3.10. From the plot it can be seen that the BER increases as $\frac{E_b}{N_o}$ decreases. A BER rate of 10^{-6} is generally considered an adequate value for digital communication, which relates to a $\frac{E_b}{N_o}$ value of roughly 10.5 dB.

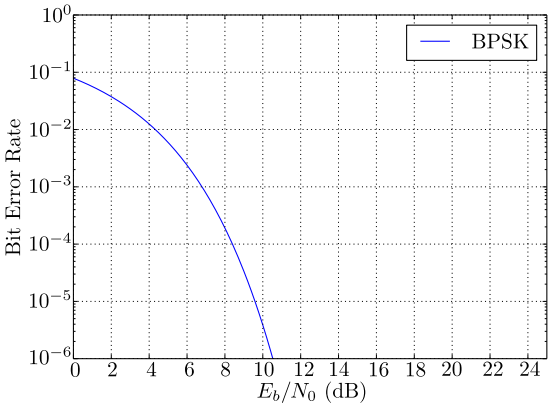


Figure 3.10 – Bit error rate for BPSK modulation.

The expected bit rate for BPSK can be calculated using Equation 3.3.8. Assuming a baud rate of 4800, the following bit rate is achieved:

$$C = R\log_2(M)$$
$$C = 4800\log_2(2)$$
$$C = 4800\ bps$$

QPSK: Quadrature Phase Shift Keying (QPSK) is a form of PSK modulation where k=2 and M=4. A coding diagram for QPSK modulation can be viewed in Figure 3.11.

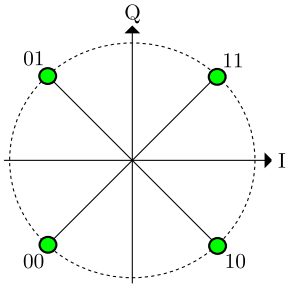


Figure 3.11 – Gray coded QPSK modulation diagram.

From the figure it can be seen that there are four waveforms available with QPSK, which lead to the 2 bits per symbol. The waveforms differs only in phase, which is separated by $\frac{\pi}{2}$

radians. QPSK consists of two BPSK signals in phase quadrature, which leads to the same BER equation as BPSK:

$$BER = \frac{1}{2}\,\text{erfc}\left(\sqrt{\frac{E_b}{N_o}}\right) \tag{3.3.10}$$

The BER graph for QPSK is the same as BPSK and can be viewed in Figure 3.12

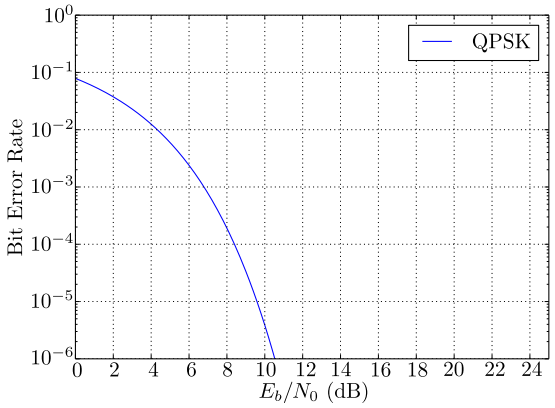


Figure 3.12 – Bit error rate for QPSK modulation.

The bit rate for QPSK with a baud rate of 4800 is as follows:

$$\begin{aligned} C &= R\log_2(M) \\ C &= 4800\log_2(4) \\ C &= 9600\ bps \end{aligned}$$

QPSK's data rate is double that of BPSK, while using the same amount of bandwidth. QPSK is a more efficient modulation technique compared to BPSK, but it is more difficult to implement.

M-ary PSK: M-ary PSK is PSK modulation where M>4, which leads to a spacing of $\frac{2\pi}{M}$ radians between each waveform. The BER for M-ary PSK is:

$$BER = \frac{1}{k}\,\text{erfc}\left(\sqrt{k\frac{E_b}{N_o}}\sin\left(\frac{\pi}{M}\right)\right) \tag{3.3.11}$$

A composite BER graph of the various PSK graphs can be viewed in Figure 3.13.

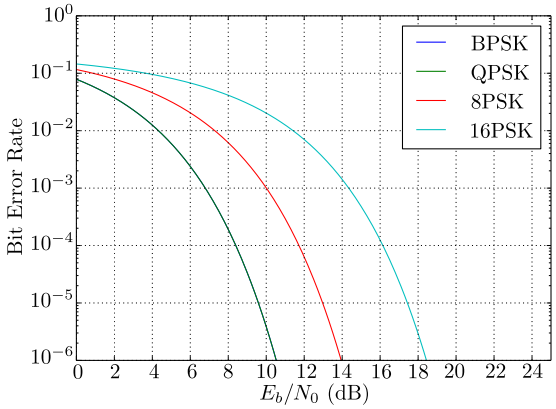


Figure 3.13 – Bit error rates for PSK modulation schemes.

The probability of data corruption increases with the number of symbols, which leads to a higher required $\frac{E_b}{N_o}$ for a specific BER. This can be visually deduced from Figure 3.13. The advantage of more symbols is an increased bit rate for the same bandwidth - as can be seen in Table 3.1.

Table 3.1 – Bit rates for PSK modulation techniques with a baud rate of 4800.

Bit(s)	Symbols	Modulation	Bit Rate
1	2	BPSK	4 800
2	4	QPSK	9 600
3	8	8PSK	14 400
4	16	16PSK	19 200

3.3.2.2 Frequency Shift Keying

The most basic form of Frequency Shift Keying (FSK) is where two different frequencies are used to represent a logical one and zero. This is known as Binary Frequency Shift Keying (BFSK) and a basic illustration is presented in Figure 3.14.

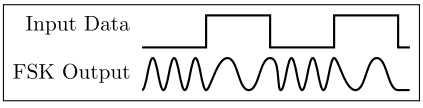


Figure 3.14 – Illustration of BFSK modulation.

These frequencies are equally spaced and are centred around the carrier frequency. There are two forms of FSK demodulation, namely coherent and non-coherent FSK. With coherent

FSK the receiver is phase-locked with the carrier signal, which yields a BER of:

$$BER = \frac{1}{2}\ \text{erfc}\left(\sqrt{\frac{1}{2}\frac{E_b}{N_o}}\right) \tag{3.3.12}$$

Non-coherent demodulation, where a phase-lock is not implemented, results in reduced performance compared to coherent modulation. The theoretical BER of non-coherent FSK can be viewed in Figure 3.15.

Minimum Shift Keying: Minimum Shift Keying (MSK) is a modulation technique similar to FSK, but the frequency changes always occur at a zero amplitude (or zero crossing) position, which eliminates discontinuities and maintains phase continuity. Signals with no discontinuities produce better spectral properties than signals with discontinuities. The BER for MSK is the same as for basic QPSK:

$$BER = \frac{1}{2}\ \text{erfc}\left(\sqrt{\frac{E_b}{N_o}}\right) \tag{3.3.13}$$

Gaussian MSK: This technique is an expansion on standard MSK with the signal passing through a Gaussian filter which reduces the size of the side-bands around the carrier frequency.

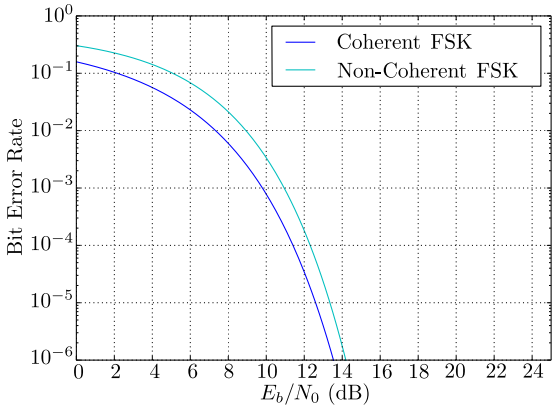


Figure 3.15 – Bit error rate for FSK modulation.

These modulation techniques have the same amount of symbols as PSK and will therefore have the same bit rates as in Table 3.1.

3.3.2.3 Amplitude Shift Keying

Amplitude Shift Keying (ASK) relies on differentiating the amplitude of the signal to encode data. Figure 3.16 illustrates a binary ASK signal. The presence of a signal indicates a logical 1 and no signal indicates a logical 0. This is a very basic modulation technique and it highly susceptible to noise and interference, which leads to poor performance [40]. It is rarely used for space communications.

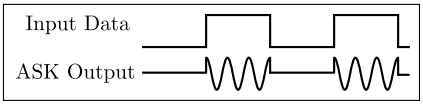


Figure 3.16 – Illustration of ASK modulation.

3.3.3 RF Channel Access Control Techniques

In most telecommunication applications, multiple devices need to make use of the same RF channel for communication, as each device (or device pair) cannot operate on its own frequencies and bandwidth. Channel access control has to be implemented in an organised fashion, in order to improve data throughput and efficiency. This section briefly discusses a few existing channel access methods.

3.3.3.1 Round Robin Polling

Round Robin Polling (RRP) is the technique of using a controlling network node to poll each other unit in the network and to request if there is data to be transmitted. This method avoids collisions, but there is some communication overhead involved with the polling process.

3.3.3.2 Frequency Division Multiple Access

With Frequency Division Multiple Access (FDMA), the available spectrum is divided among all the devices on the network. Each device has access to its own dedicated spectrum. FDMA can be applied for analogue or digital signals [41].

3.3.3.3 Time Division Multiple Access

Time division Multiple Access (TDMA) operates in the time domain and it can only be used for digital network connectivity [42]. The use of the RF channel is divided into timed slots where each device has a designated slot. Each device has exclusive access to the channel during its assigned slot.

3.3.3.4 Code Division Multiple Access / Spread Spectrum

Code Division Multiple Access (CDMA), also know as 'spread spectrum', modulates a signal onto a Pseudo Random Noise (PRN) signal, thus spreading the data signal over the available bandwidth. All the devices within a network share the same bandwidth and the receiver demodulates the signals by means of auto-correlation with the PRN code. Unwanted signals will not correlate with the PRN code and will appear as white noise in the signal [41]. This modulation technique allows the simultaneous communication of multiple device pairs on the same frequency and bandwidth. The noise generated by other devices is much smaller than with other modulation techniques, since the signal is spread over a larger bandwidth. A generic diagram for spread spectrum modulation can be viewed in Figure 3.17.

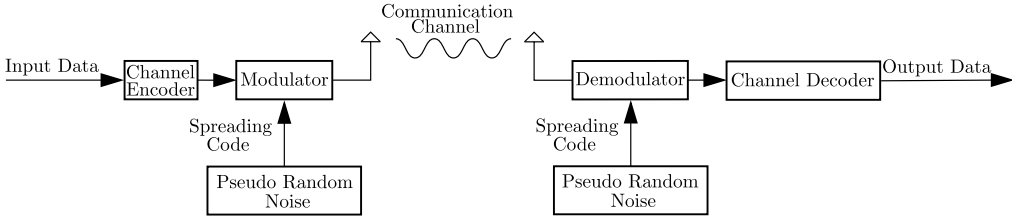


Figure 3.17 – Generic spread spectrum signal modulation. Adapted from Ref. [4].

3.3.4 Antennas

An antenna is a another critical part to a radio communication network. The antenna transmits and receives signals to facilitate communication.

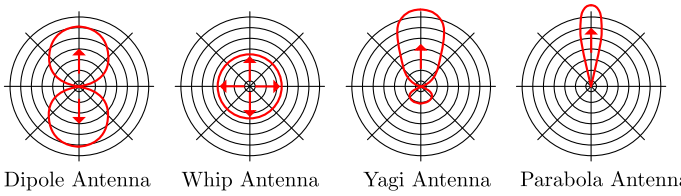


Figure 3.18 – General radiation patterns of the most common antenna types [5].

Figure 3.18 illustrate the generic radiation patterns of some common antenna types. Some of the antennas are non-directional as they radiate omni-directionally and usually provide low signal gain. The directional antennas have more focused beams which give larger signal gains, but these antennas have to be placed in specific orientations to transmit and receive effectively. Some of the directional antennas are never used for nano-satellites due to their stringent pointing accuracy requirements, physical size and weight.

Monopole and Dipole antennas are mostly used for CubeSat communications, which rely on Ultra High Frequency (UHF) or Very High Frequency (VHF) communication. When the 2.4 GHz spectrum is used, a patch antenna is utilised because of its small size and weight. Patch antennas are cost effective and can be mounted on a flat surface, which conserves space and complexity as no deployment mechanism is required.

3.3.4.1 Antenna Map

In the simulator, antenna gain patterns are represented as antenna gain maps, as shown in Figure 3.19. An antenna map is attached to every transmitter and receiver in the simulation. The elevation and azimuth is provided to the antenna so that the gain at that specific point can be retrieved. An elevation of $0°$ is parallel to the surface of the earth with $90°$ (away from earth) and $-90°$ (towards earth) being perpendicular to the surface of the earth. Azimuth is measured from the North where an azimuth angle of $0°$ points exactly North.

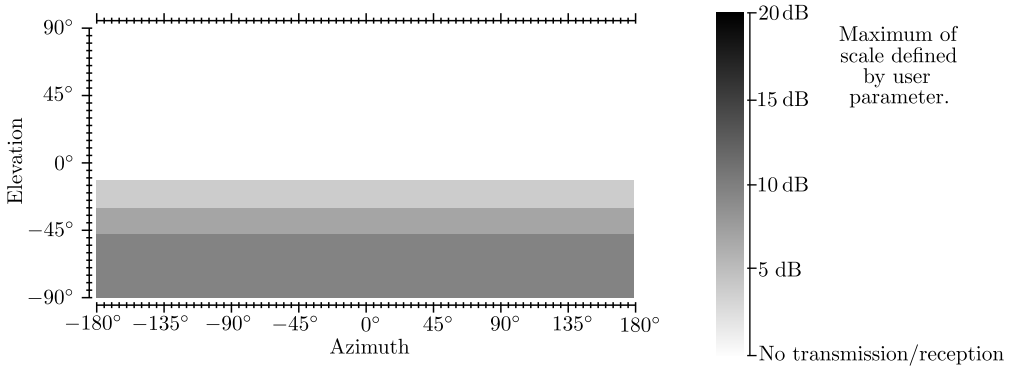


Figure 3.19 – Example of an antenna map for a satellite-to-earth transmission antenna.

The antenna pattern is static for all objects in the simulation, including moving objects, as satellite orientation and ground station antenna tracking are not modelled. The antenna patterns must be designed to assume that the antenna will always point in the desired direction. This can be achieved by creating a pattern with a single gain value across a section, or with a single gain value across all elevation and azimuth angles.

The antenna maps are similar to the elevation maps described in Section 3.2.3.1. Using a visual gain representation makes it easier for a user to implement a custom gain map. The drawback with this implementation is the limitations associated with the PNG bit map format. The colour intensity value is limited to a maximum of 255 (8 bits), which limits the resolution of the antenna gain. The maximum antenna gain is a user specified value, which allows for some flexibility in the antenna design. The current solution is adequate for this project, but an improved implementation may be required for future projects.

It would be more convenient if an antenna map can be imported directly from antenna simulation software such as FEKO [43]. This will provide a more accurate way of simulating certain antenna models, but it is uncertain if this is technically feasible. It is also not possible to implement negative dB antenna gain with the PNG image representation. Negative antenna gain can be simulated, by adding an offset to the RF channel model, but a future update can re-scale the image gain map to include negative values.

3.4 Link Budget and Error Modelling

This section discusses the design of the link budget and how bit errors are modelled in the simulation. The link budget is used to determine the BER and the bit errors are then generated at the specific BER in the simulation.

3.4.1 Link Budget Design

A relationship between the various characteristics of a RF link has to be defined, to determine the strength of the received signal. This is done with a link equation or link budget. All the parameters of a communication link are summarised in one equation to determine the $\frac{E_b}{N_0}$, which determines how strong the received signal is. This equation is formulated as follows [3]:

$$\frac{E_b}{N_0} = \frac{PL_lG_tL_sL_aG_r}{kT_sR} \tag{3.4.1}$$

where, $\frac{E_b}{N_0}$ the ratio of received energy-per-bit to noise-density,
P the transmitted power (W),
L_l the transmitter-to-antenna line loss,
G_t the transmitter antenna gain,
L_s the space loss,
L_a the transmission path loss,
G_r the receiver antenna gain,
k the Boltzmann constant,
T_s the system noise temperature (K),
R the data rate (bps).

The link equation is a product of successive terms and can therefore be expressed in terms of decibels (dB).

$$\begin{aligned}\frac{E_b}{N_0} &= P_{dB} + L_{l\ dB} + G_{t\ dB} + L_{s\ dB} + L_{a\ dB} + G_{r\ dB} - 10\log(k) - 10\log(T_s) - 10\log(R)\\ \frac{E_b}{N_0} &= P_{dB} + L_{l\ dB} + G_{t\ dB} + L_{s\ dB} + L_{a\ dB} + G_{r\ dB} + 228.6 - 10\log(T_s) - 10\log(R)\end{aligned} \tag{3.4.2}$$

In equation 3.4.2, T_s is in Kelvin (K) and R in bps. All the various terms of the link budget are now required. The transmitted power (P), the transmitter antenna gain (G_t), the receiver antenna gain (G_r), the space loss (L_s) and the data rate (R) are values that will be extracted from the simulation environment at the point and time of calculation. T_s, L_l and L_a can be derived prior to the simulation as long as transmission frequencies are not set to unreasonably high values.

3.4.1.1 Free Space Loss

The free space loss (also known as the spreading loss) can be calculated as follows [3]:

$$L_s = \left(\frac{\lambda}{4\pi D}\right)^2 \tag{3.4.3}$$

where, λ the signal wavelength in meters,
D the propagation distance in meters.

However, it is known that the relationship between the wavelength and frequency is:

$$\lambda = \frac{c}{f} \tag{3.4.4}$$

where, c the speed of light,
f the signal frequency in Hz.

If Equation 3.4.4 is substituted into Equation 3.4.3, we get another equation for free space loss:

$$L_s = \left(\frac{c}{4\pi D f}\right)^2 \tag{3.4.5}$$

Converting Equation 3.4.5 to dB gives the following:

$$\begin{aligned} L_{s\ dB} &= 20\log(3\times 10^8) - 20\log(4\pi) - 20\log(D) - 20\log(f) \\ &= 147.55823 - 20\log(D) - 20\log(f) \end{aligned} \tag{3.4.6}$$

3.4.1.2 Atmospheric Attenuation

As a signal passes through the Earth's atmosphere it experiences attenuation due to the atmospheric gasses. Figure 3.20 shows the total zenith attenuation experienced due to atmospheric gasses.

The International Telecommunications Union (ITU) provides a recommendation on calculating the signal attenuation due the atmospheric gasses [6]. The recommendation contains two approaches to calculate the attenuation: an advanced approach and a simple approach. However, the latter is only applicable to altitudes of about 10 km as the attenuation in that case, is estimated using curve fitting.

The effect of atmospheric attenuation at low frequencies will be negligible compared to other sources of attenuation. The simulations for this project make use of the sub-gigahertz range of frequencies. Thus, atmospheric attenuation will be very small and will be excluded from the current link budget ($L_{a\ dB} = 0$ dB). Future work can include the implementation of the ITU atmospheric attenuation model [6] for simulations that need to make use of frequencies above 10 GHz.

3.4.1.3 Weather Related Attenuation and Noise

Rain, clouds and fog have an influence on communication links, especially at higher frequencies. These elements are difficult to model with a dynamic link budget, where satellites

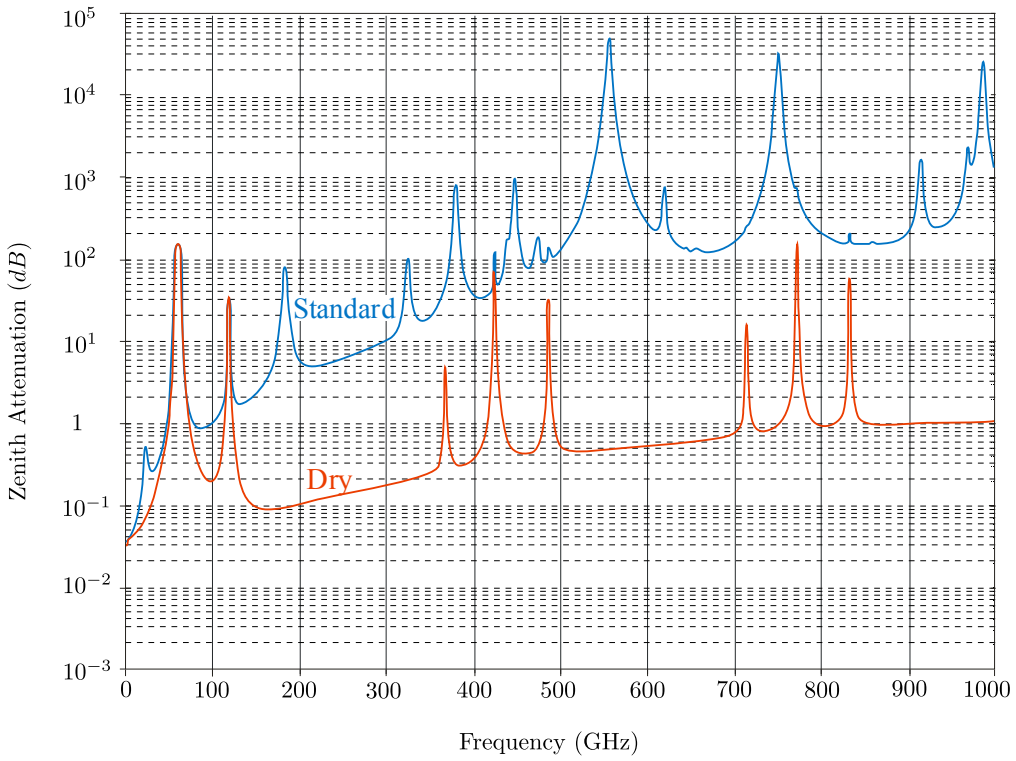


Figure 3.20 – Total zenith attenuation due to atmospheric gases. Adapted from Ref. [6].

move around the earth. It will be assumed for this project that the weather is clear at all times ($L_{a\ dB}$ remains 0 dB).

The implementation of a weather model is a huge endeavour, since the simulation is on a global scale. The earth is divided into different weather regions with varying characteristics depending on the time of year. This simulation element will form part of future work as it is not of particular interest for this project.

3.4.1.4 Noise Temperature

It is challenging to model system noise temperature for a dynamic simulation environment. Noise temperature elements depend on the position, altitude and elevation of the sender and receiver. Assumptions will be made regarding the noise temperature to simplify the model. A complex model will require in-depth channel modelling and the verification of such a model could take some time. This is unfortunately beyond the scope of this project.

The first noise temperature introduced into the system is the noise from the sky (T_{SKY}), which varies from 2 K to 90 K depending on the elevation. A sky noise temperature graph can be viewed in Figure 3.21.

Since the simulator will mostly be used to test below the 5 GHz frequency range, it will assume a value of 40 K for T_{SKY} when assuming a worst case elevation angle of 2° to 3°.

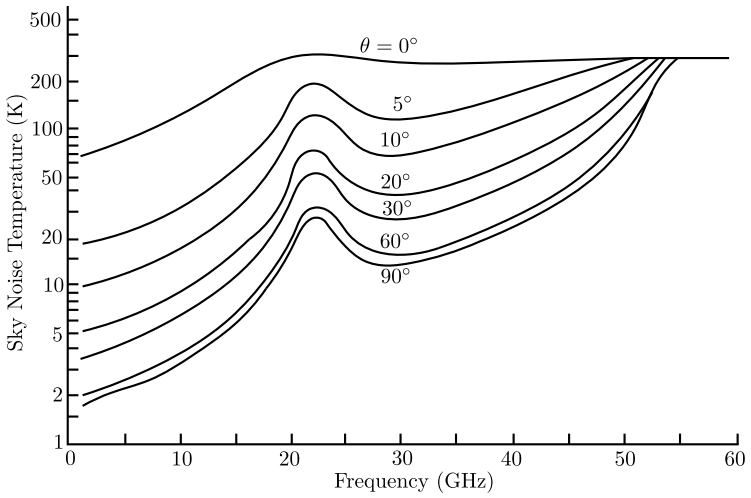


Figure 3.21 – Sky noise temperature for clear air and 7.5 g/m^3 of water vapour with elevation angle θ. Adapted from Ref. [7].

Figure 3.21 is only applicable to ground-to-space communication and it is more challenging to estimate the noise temperatures between inter-satellite links.

The other noise sources are the galactic noise ($T_{GALACTIC}$) and the noise from the earth (T_{EARTH}). $T_{GALACTIC}$ is the background cosmic noise and is assumed to be 2.7 K for all links [7][44]. T_{EARTH} is the noise generated by the earth and it is assumed to be 60 K, as its value is typically between 10 K and 100 K [7].

The antenna temperature (T_A) is then:

$$\begin{aligned} T_A &= T_{SKY} + T_{GALACTIC} + T_{EARTH} \\ &= 40 + 2.7 + 60 \\ &= 102.7\ K \end{aligned}$$

A typical amplifier has a noise figure (F) of between 2 dB and 3 dB [44]. A value of 3 dB will be used for this link budget. A future option can include a user selectable noise figure. The chosen noise figure can be converted to a noise temperature as follows:

$$\begin{aligned} T_R &= T_0(F-1) \\ &= 290(10^{3/10}-1) \\ &= 288.626\ K \end{aligned} \tag{3.4.7}$$

The physical receiver temperature (T_{FRX}) is assumed to be 293.15 K (20°C) and the losses between the receiving antenna and the receiver input (L_{FRX}) is 0.5 dB. The system noise

temperature (T_S) can now be calculated as:

$$\begin{aligned} T_S &= \frac{T_A}{L_{FRX}} + T_{FRX}\left(1 - \frac{1}{L_{FRX}}\right) + T_R \\ &= \frac{102.7}{10^{0.5/10}} + 293.15\left(1 - \frac{1}{10^{0.5/10}}\right) + 288.626 \\ &= 412.037\ K \end{aligned} \tag{3.4.8}$$

3.4.1.5 Total Link Equation

The transmitter line loss ($L_{l\ dB}$) has a typical value of -0.5 dB across all frequency ranges [3]. Some additional losses are also added at this stage. These include an implementation loss of -2 dB and polarisation losses of -3 dB [3][7]. This will be specified as additional losses (L_{add} = -5 dB). The total link equation for the simulation can now be calculated from Equation 3.4.2:

$$\begin{aligned} \frac{E_b}{N_0} &= P_{dB} + L_{l\ dB} + G_{t\ dB} + L_{s\ dB} \\ &\quad + L_{a\ dB} + G_{r\ dB} + 228.6 - 10\log(T_S) - 10\log(R) + L_{add} \\ &= 10\log(P) - 0.5 + G_{t\ dB} + 147.558 - 20\log(D) - 20\log(f) \\ &\quad - 0 + G_{r\ dB} + 228.6 - 10\log(412.037) - 10\log(R) - 5 \\ &= 10\log(P) + G_{t\ dB} + G_{r\ dB} - 20\log(D) - 20\log(f) - 10\log(R) + 344.5086 \end{aligned} \tag{3.4.9}$$

$P(watt)$, G_t(dB), G_r(dB), D(meters), f(Hz) and R(bps) are values that will be extracted from the simulation environment at the point and time of calculation. In the simulation, equation 3.4.9 will be implemented for all links to determine the $\frac{E_b}{N_0}$ and, in turn, the BER.

3.4.1.6 Verification

A simple way to verify a link budget is to compare the designed link budget to the link budgets proposed in other publications. This will instill confidence in the design of the simulation link budget. Some of the identified publications did not state all the link budget variables clearly. Certain link budget variables, especially propagation distances, had to be derived from the publications' contents. The link budget comparisons can be viewed in Table 3.2.

In some cases, the designed link budget was slightly optimistic compared to other publications' budgets. The 2 dB to 3 dB difference can be attributed to the lack of rain attenuation, as explained previously. The large variation in the UHF link of the LFR mission is due to the difference in system noise temperature in the publication, which contributes about -5 dB of the difference.

Link budget design is not an exact science and contains numerous estimates. Two link budgets will rarely be exactly the same. The variations observed in Table 3.2 is therefore to be expected. Overall, the link budget calculation is close to what is expected and is adequate for simulation purposes.

Table 3.2 – Link budget comparison with various publications.

Mission	Frequency (MHz)	Publication $\frac{E_b}{N_0}$ (dB)	Simulation $\frac{E_b}{N_0}$ (dB)	Difference (dB)
OuterNet [45]	150	19.26	21.377	2.117
	450	12.7	14.814	2.114
	150	28.67	33.138	4.468
	450	18.13	22.595	4.465
	150	22.25	24.367	2.117
	450	12.71	14.824	2.114
	150	22.51	26.128	3.618
	450	12.97	16.585	3.615
	2200	14.46	16.575	2.115
	2200	17.54	16.575	-0.9647
SOLARA [45]	2440	13.69	14.517	0.827
	2440	14.61	17.52	2.917
LRF Mission [46]	145	33.498	40.377	-0.413
	450	40.37	33.360	-7.016
SMAD [3]	2000	45.5	45.749	0.249
Mission	**Frequency (MHz)**	**Publication $\frac{C}{N_0}$ (dB.Hz)**	**Simulation $\frac{C}{N_0}$ (dB.Hz)**	**Difference (dB.Hz)**
ICC Paper [47]	2000	76.65	73.41	-3.23

3.4.1.7 Potential Future Improvements

Weather Models - There is a need for a weather model, as clear weather is currently assumed for all communication links. Rain, clouds and fog are all sources of additional noise and attenuation. Rain, for example, has a large influence on satellite communication and can cause periods of communication outages.

Atmospheric Models - Currently the simulation ignores atmospheric attenuation due to its small effect at lower frequencies. If higher frequencies are modelled in future simulations, an atmospheric model will be required. The ITU provides a recommendation for calculating the signal attenuation due the atmospheric gasses [6]. The attenuation recommendation is in line with Ref. [48], which contains the reference standard atmospheric data required to calculate the attenuation of gasses.

Accurate Link Budget For Different Links - The current link budget makes assumptions in order to simplify the scope of this project, but if vastly different link designs are used, different link budget models may be required. A different model should technically be used for inter-satellite links and ground-to-satellite links. The difficulty with this task is to accurately predict the noise temperatures involved with each link. The presence of the sun, moon and atmosphere will also influence the noise temperature and will affect the quality of a link.

3.4.2 Error Modelling

Now that the link budget is available, it is possible to calculate the $\frac{E_b}{N_0}$ and then calculate the theoretical BER for the specific modulation schemes, as discussed in Section 3.3.2. The BER value is then applied to a data frame to determine how many bit errors occurred during the transmission of a frame.

3.4.2.1 Binary Symmetric Channel

The propagation channel in SatSim is modelled as a binary symmetric channel [49]. A binary symmetric channel is a model for a memoryless probability channel. It is often used to model an event with two possible outcomes, where past events have no influence on the probability of current events. In this case, the binary symmetric channel is applied to the reception of bits. The probability of a transmitter transmitting a 1 and a receiver receiving a 0 is:

$$P(1|0) = P_e$$

where P_e is the probability of receiving an erroneous bit. This probability also holds true for:

$$P(0|1) = P_e$$

And the probability for receiving the correct bits are:

$$P(0|0) = P(1|1) = 1 - P_e$$

as the total probability should equal 1. A visual definition of a binary symmetric channel can be viewed in Figure 3.22.

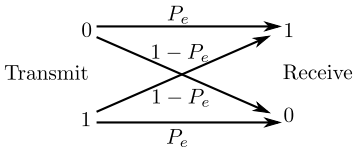


Figure 3.22 – Binary symmetric channel.

When the binary symmetric channel model is applied, the probability of bit errors can be written as follows:

$$P_{\text{bit error}} = BER$$
$$P_{\text{no error}} = 1 - BER$$

3.4.2.2 Bit Error Calculation Methods

Since a network frame can contain thousands of bits, it is wise to investigate different methods of generating bit errors in a transmitted frame. Four methods of generating bit errors in a frame were investigated:

1. A single process approach, where error generation is not shared across multiple processor cores. A frame is processed bit-by-bit where a random number is generated for each bit in a frame and compared to the BER probability, to determine if a bit error has occurred. This is used as a baseline as it is the most straight forward approach.
2. A custom multiprocessing approach, similar to the previous method, but the bit error generation is spread across multiple processes. This involves using the multiprocessing libraries in Python to divide the calculations amongst multiple processor cores and returning the information back to the main process. If a frame consists of 1000 bits, each core on a 4-core processor will do a bit error check for only 250 bits.
3. A Python multiprocessing pool approach. This is similar to the previous method, but it is structured slightly differently, due to the use of different Python libraries.
4. A custom multiprocessing approach, where Bernoulli trials are used to reduce the total amount of random number generations. This method should require the generation of only one random number per frame. The potential drawback is that the calculation of the Bernoulli trial is computationally expensive, even with optimisations, as it requires the calculation of large factorials.

The first three approaches are practically the same, as they involve generating a random number for each bit and then comparing it to the BER probability. Only the implementation of these three approaches differ. The Bernoulli method is more complex as it involves some probability theory.

The performance index that would compare all the bit error calculation methods, is the time it takes to determine the bit errors for the same test configuration. Bernoulli trials and probability distributions will also be used to verify the results achieved with the first three methods.

3.4.2.3 Bernoulli Trials

Since a bit error can be seen as an event that has only two outcomes (it can either occur, or not) it can be approached as a Bernoulli trial. A Bernoulli trial is an event where the following holds true:

- The probability of an event occurring is p,
- The probability of the event not occurring is $q = 1 - p$,
- The probability of the current event is not influenced by the probabilities of previous events.

A Bernoulli trial is formulated as follows [37]:

$$P(k \text{ events out of } n \text{ trials}) = \binom{n}{k} p^k (1-p)^{n-k}$$

with,

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

which gives,

$$P(k \text{ events out of } n \text{ trials}) = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k} \tag{3.4.10}$$

When the probability of a bit error is applied to Equation 3.4.10 it can be rephrased as follows:

$$P(k \text{ bit errors out of } n \text{ bits}) = \frac{n!}{k!(n-k)!} p^k (1-p)^{n-k} \tag{3.4.11}$$

Equation 3.4.11 only yields the probability of a fixed amount of bit errors. A probability distribution has to be implemented to calculate the probability of packet failure.

It is possible to calculate the probability of x or more bit errors with a probability distribution. The value of x can be any real value, as long as it is less than the total amount of bits in a transmitted frame. The probability distribution is generated by calculating the probability of all the combinations of bit errors. The probability distribution for a BER of 10^{-2} and a frame consisting of 256 bits can be viewed in Figure 3.23.

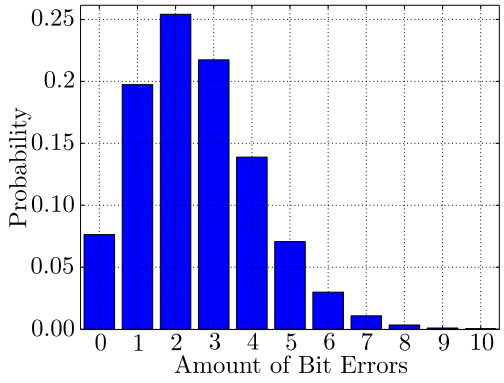


Figure 3.23 – Calculated probability distribution for a $10^{-2} BER$. This graph only illustrates the number of bit errors from 0 to 10, as the probability of more errors is minute and not visible if added to the graph.

A first observation is that the probability of 10 or more bit errors in a 256 bit frame is minuscule. The bit error probability peaks at 2 bit errors. This can be confirmed theoretically

as 1 out of every 100 bits will be corrupted with the given BER, which in the case of a 256 bit frame will be 2.56 bits per 256 bit frame.

The distribution in Figure 3.23 can now be used to calculate, for example, the probability of 3 or more bit errors, by summing the probability of all the bit errors, which are greater or equal to 3. However, this will involve summing 253 probability values, which lead to a probability of 0.4722. To reduce the amount of computation, the probability can be inverted and subtracted from 1. Instead of summing all the probabilities applicable to 3 bit errors, the probabilities applicable to less than 3 bit errors can be added and subtracted from 1. The probabilities applicable to less than 3 bit errors sum to 0.5278. If this is subtracted from 1, an answer of 0.4722 is calculated (remembering that $q = 1 - p$).

This provides the same result, but with less computational effort as only 4 additions had to be made instead of 253. There is also no need to calculate the entire distribution function as not all values are summed.

3.4.2.4 Bit Error Performance Calculation Comparison

The first performance tests were done on a single frame. The performance tests were conducted with a BER of 10^{-4} over various frame sizes. The results can be viewed in Table 3.3.

The slowest method is the pool-processing method. The execution time for this method is very consistent, but this can be attributed to the computational overhead involved with process management, compared to the actual execution time of the work required.

The Bernoulli method is also slow and its execution time does not scale well with the size of the frame being calculated. The whole process involves only two Bernoulli calculations, since the probability of 0 and 1 bit error are calculated. Each Bernoulli calculation is done on a separate core. The extensive execution times are due to the increasing size of the factorial calculations required by the various frame sizes.

The multiprocessing method yielded the second fastest results. Again, the slow processing time can be attributed to the overhead involved with setting up multiple threads.

The single process approach is the fastest by some margin. It scales quite well with the tested frame sizes. It can be assumed that the single process will eventually become slower than the multiprocessing method, but it would require massive frame sizes, which does not occur frequently in satellite communications.

The single process method is implemented in SatSim, not only due to its obvious processing speed advantage, but also the simpler implementation when working within a single process. The performance is one feature to keep in mind, but the statistical validity of such a process needs to be proved. This can be achieved by comparing results with a probability density distribution. The probability density for a BER of 10^{-4} and frame size of 1024×8 is shown in Figure 3.24.

From the probability density distribution it can be calculated that the probability of losing a frame is 0.1981222 (the probability of having more than 1 bit error in a frame). If 100 000

Table 3.3 – Performance comparison of bit error calculation methods. Bernoulli method only calculates the probability of two or more bit errors. Computed on an Intel Core i5-3570 processor with four processing cores clocked at 3.40 GHz.

	Execution Time for Various Frame Sizes (ms)				
Method	1 Mb	2 Mb	3 Mb	4 Mb	5 Mb
Custom Multiprocessing	7.5	8	8	9	9
Pool Multiprocessing	102	102	102	102	102
Single Process	0.9	1.7	2.5	3.4	4.3
Bernoulli Multiprocessing	31	101	230	413	650

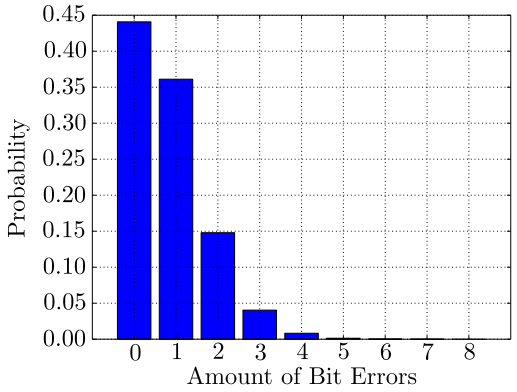


Figure 3.24 – Probability Distribution for a BER of 10^{-4} and a frame size of 1024×8. This graph only illustrates the amount of bit errors from 0 to 8 as the probability of more errors is minute and not visible if added to the graph.

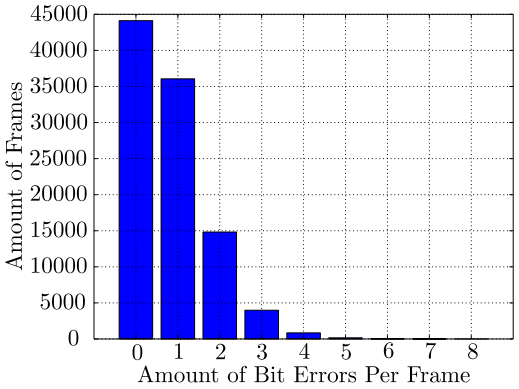


Figure 3.25 – The BER distribution to test the statistical validity of the simulation. A total amount of 100 000 frames of 1024×8 bits each with a BER of 10^{-4} was tested.

frames are transmitted, we can expect to lose 19 812.22 frames on average. The expected amount of bit errors per frame is 0.8192.

The test was configured on the same parameters and 100 000 frames were tested with the chosen BER of 10^{-4}. A frame with more than 1 bit error was deemed lost, as it was assumed the FEC can correct only 1 bit. The test generated the distribution in Figure 3.25.

The test resulted in an average amount of 0.8179 bit errors per frame, which is close to the 0.8192 that was expected. The probability distribution is also similar to that of Figure 3.24. The total amount of lost frames added up to 19 862 frames, which is again close to the expected value of 19 812.22.

Another test was conducted with 10 million frames. It resulted in 0.819217 bit errors per frame and a total of 1 980 16 lost frames (expected 1 981 222), which is only 0.053% less than expected.

It is at this point safe to assume that the method employed to model bit errors during the simulation is statistically valid.

3.4.2.5 Potential Future Improvements

Burst Errors - A burst error is when multiple bit errors occur consecutively in a transmitted frame. Burst errors are a common occurrence in the space environment and the simulator will benefit from such a model. Burst errors are one of the more common reasons for frame transmission failures in the space environment. Models for burst error have been implemented in other simulators, for example ns3 [50].

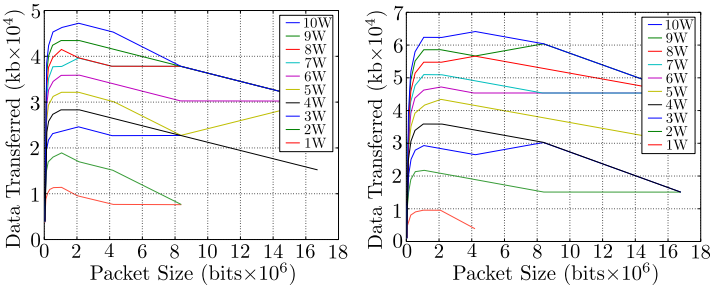
3.5 Simulation Verification

It is required to compare aspects of SatSim with previous work, in order to verify the design and implementation of the simulator. The link budget has already been verified in Section 3.4.1.6. The simulation structure, results and performance will be examined and verified in this section.

3.5.1 Approach

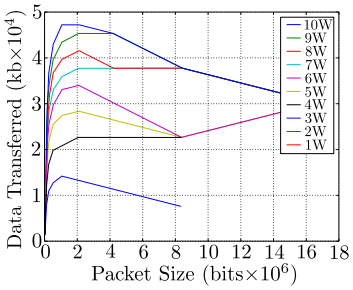
One verification approach is to duplicate simulations on multiple simulators and compare the results. This is a time intensive task, as most simulators have steep learning curves. A more achievable approach was taken where simulation configurations from other publications were duplicated in SatSim and the results compared analytically. This circumvents the requirement of implementing simulations on other simulation tools. The thesis by Bezuidenhout [8] was chosen as it contains simple tests in single satellite pass situations, with varying transmission powers and data packet sizes. The custom protocol used by Bezuidenhout is also relatively simple to implement, but not trivial. The simulations done by Bezuidenhout were conducted in OPNET [29].

The purpose of this exercise was to determine if the current implementation yields similar results and behaviour. The link budget and simulation logic could be deemed accurate if similar results are achieved. Exact same results are not expected, but similar behaviour is. Simulation results of Bezuidenhout can be viewed in Figure 3.26.



(a) BPSK at 76 kbps.

(b) QPSK at 152 kbps.



(c) 8PSK at 228 kbps.

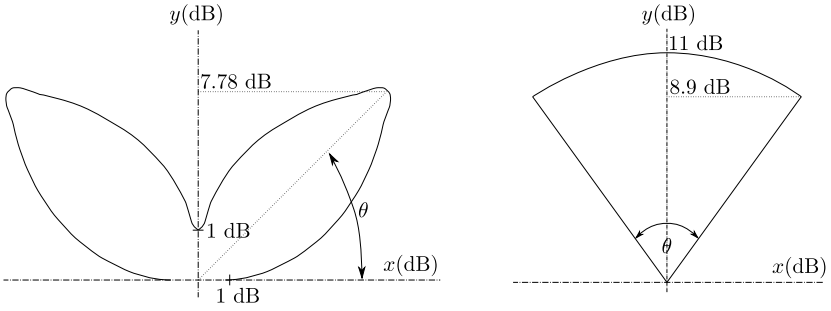
Figure 3.26 – OPNET simulation results with BCH10% coding [8].

3.5.2 Simulation Configuration

The test configuration consists of the following:

- A single satellite and single ground station.
- Satellite is at an altitude of 500 km.
- Simulate a single satellite pass over the ground station (roughly 15 minutes).
- Uplink frequency of 2400-2400.1 MHz.
- Downlink frequency of 2400.2-2400.3 MHz.
- Ground station continuously transmits packets at 76 kBaud.
- Use BCH FEC coding, which is able to correct 10% of bit errors. More than 10% errors leads to a frame being discarded.
- Simulate for packet sizes varying from 1 kb, 2 kb, 4 kb, ... 16 Mb.
- Simulate for transmission power varying from 1 W, 2 W, 3 W, ... 10 W.
- Satellite continuously transmits at 10 W.

The antennas used for the ground station and the satellite can be viewed in Figures 3.27(a) and 3.27(b) respectively. The satellite antenna is assumed to be pointing nadir at all times, although this is not specifically stated in the original simulation description.



(a) Ground station antenna ($\theta = 45°$). (b) Satellite antenna ($\theta = 72°$).

Figure 3.27 – Ground and satellite antennas. Adapted from Ref. [8].

The test protocol consists of four types of packets [8]. The packet structures can be viewed in Figure 3.28.

Data	Source Address (32)	Destination Address (32)	Time Stamp (21)	CNT (10)	ACK (1)	From Satellite (1)	Data (D)	FEC (F)
ACK	Source Address (32)	Destination Address (32)	Time Stamp (21)	CNT (10)	ACK (1)	From Satellite (1)	FEC (F)	
RTS	Source Address (32)	RTS (1)	Time Busy (9)	CNT (10)	From Satellite (1)			
CTS	Host Address (32)	CTS (1)	Time Busy (9)	CNT (10)	From Satellite (1)			

Figure 3.28 – Protocol packet structures. Adapted from Ref. [8].

Some of the fields were implicitly used in the simulation. For example, the 'From Satellite' field was not required for a single ground station and satellite configuration.

The data size (D) of the data packet was defined as:

$$D = (\text{packet size}) - 256 - F$$

For a packet size of 1024 bits the data size would be::

$$D = 1024 - 256 - F$$

The 256 bits were defined as the sum of the header bits, which include the Source Address, Destination Address, Time Stamp etc. and the F bits are the FEC bits. However, the 256 bits appeared to be incorrect, as the header information added up to 97 bits. It is not known if this was done for a specific reason, but the 256 bits were maintained for the sake of achieving similar results.

The FEC scheme applied in the test protocol was not clearly defined by Bezuidenhout. The author stated that BCH coding is used, which can correct 10% of packet errors, but the specific implementation is not defined. A code of BCH(511, 157, 51) was implemented in the SatSim simulation to mimic the 10% error correction as closely as possible. This code has a block length of 511 bits, 157 data bits and can correct 51 bits. This results in a coding rate of roughly 0.3 and the ability to correct nearly 10% of bit errors. A summary of other possible BCH codes can be viewed in Table B.1.

The behaviour of the protocol was also not clearly defined. The use of Request To Send (RTS) and Clear To Send (CTS) packets imply there was a form of broadcasting involved. For the initial SatSim simulations it was assumed that a link is established after a CTS packet is received. Data packet transmission will then commence until the link is broken. Then an RTS packet will be broadcasted again to establish a new link. Figure 3.29 demonstrates how the protocol establishes and re-transmits a data packet if a packet is corrupted. The behaviour of the protocol varies, depending on the chosen time-out periods for RTS, CTS and DATA packets.

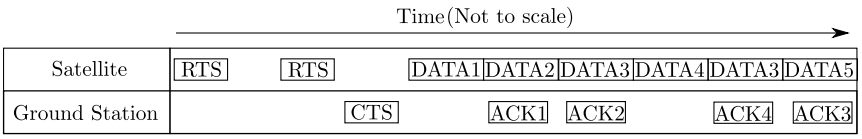


Figure 3.29 – Assumed protocol behaviour.

3.5.3 Simulation Results and Comparison

Initial simulations did not yield similar results. After numerous modifications to the simulation configuration, the results started to show similar behaviour to those in Figure 3.26. The modifications included:

- The satellite antenna in Figure 3.27(b) was initially assumed to not broadcast outside the 72° region. This was changed to a 0 dB region from −90° to 90° (outside the 72° region). This corrected a situation where all packet sizes and powers provided nearly the same data throughput, as the link budget was within margin as soon as the ground station was within boresight.
- The 'Data Transferred' on the y-axes of the Figures in 3.26 were deemed to be the count of all successful packet transfers, which included the reception of RTS, CTS, ACK and data packets with successful ACK messages. The entire data packet was counted and not only data payload inside the packet. This explains the lack of detail regarding the FEC codes, since it is not required to calculate the data payload size from each packet.

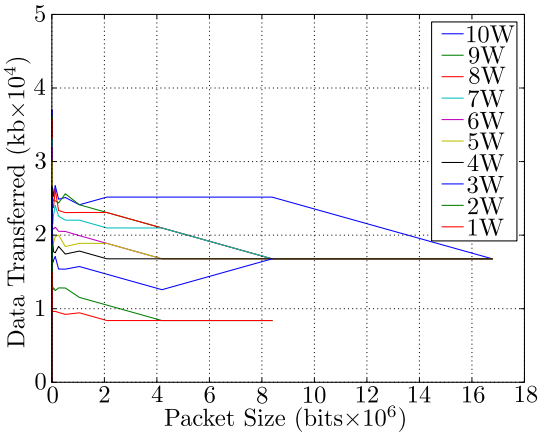


Figure 3.30 – Initial simulation with BPSK modulation with a single RTS/CTS packet combination to establish a link.

The results of the first simulation run after the modifications can be viewed in Figure 3.30. There are two noticeable differences to the expected results in Figure 3.26. Firstly, the data

transferred is about half of what is expected and secondly, the same behaviour did not occur in the results for the smaller packet sizes.

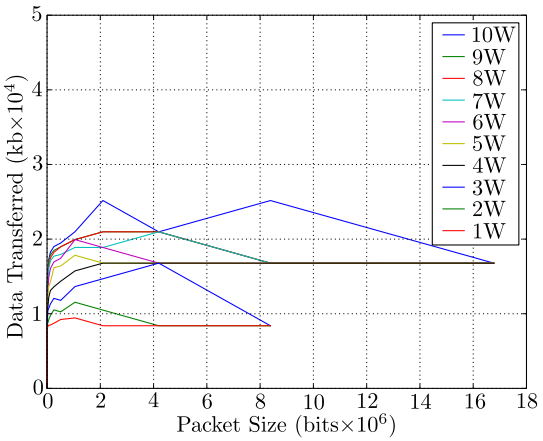


Figure 3.31 – Second simulation with BPSK modulation and modified protocol with a RTS/CTS packet combination per data packet.

The behaviour of the smaller packet sizes could be due to the protocol implementation. Since only one RTS and CTS packet combination is required to establish a link, the data transferred should be quite similar over all packet sizes. If the protocol sends an RTS and CTS packet combination per data packet, it should produce a slope as in Figure 3.26. This would occur due to the added overhead with smaller packet sizes. Figure 3.31 is the results with the modified protocol.

The new simulation, with the modified protocol, displays the curve at the smaller packet sizes, with the peaks around the 2×10^4 bits packet size. A initial conclusion would be that this is definitely a less optimised protocol than in the first case. The remaining problem is the significantly reduced data transfer values in the simulation. An initial check is to determine if the reported data transfer amounts are theoretically possible. Table 3.4 contains an analysis of the theoretical maximum data throughput.

Table 3.4 – Theoretical data throughput from 500 km altitude. Time in view is taken from Ref. [3].

Elevation (°)	**Time in View (500 km Altitude)**	**Data Transfer (at 76 kbps)**
0	11.55 min | 693.0 sec	5.267×10^4 kb
5	9.21 min | 552.6 sec	4.200×10^4 kb
10	7.38 min | 442.8 sec	3.365×10^4 kb
20	4.93 min | 295.8 sec	2.248×10^4 kb

From Table 3.4 it can be seen that it is indeed possible to get a total data transfer of above 4×10^4 kb as reported in Figure 3.26. However, this is only possible if data transmission starts at 5° elevation or lower. The simulation logs indicated that with 8W transmission power, the link will only start to function at an elevation of 25°. With 4W transmission power the link is only established at 37°. The amount of data transferred observed in Figure 3.31 seems to be in line with what is theoretically expected.

The reduced data transfer seems to be associated with the point at which the link is within margin. It appears the link is not established soon enough. This could be due to multiple reasons, including the link budget and the antenna designs. The link budget has been verified in Section 3.4.1.6 and was deemed to be adequate. The antenna design is the only remaining factor. The antenna design could have been documented incorrectly as the link is struggling to establish at low elevations. At low elevations, the ground antenna has a very low gain and the satellite antenna has a 0 dB gain, which will vastly reduce the link performance.

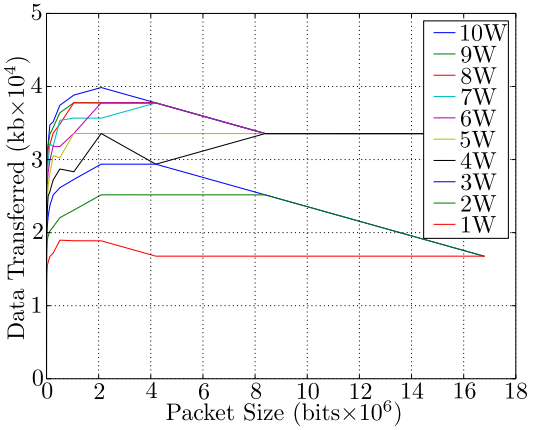
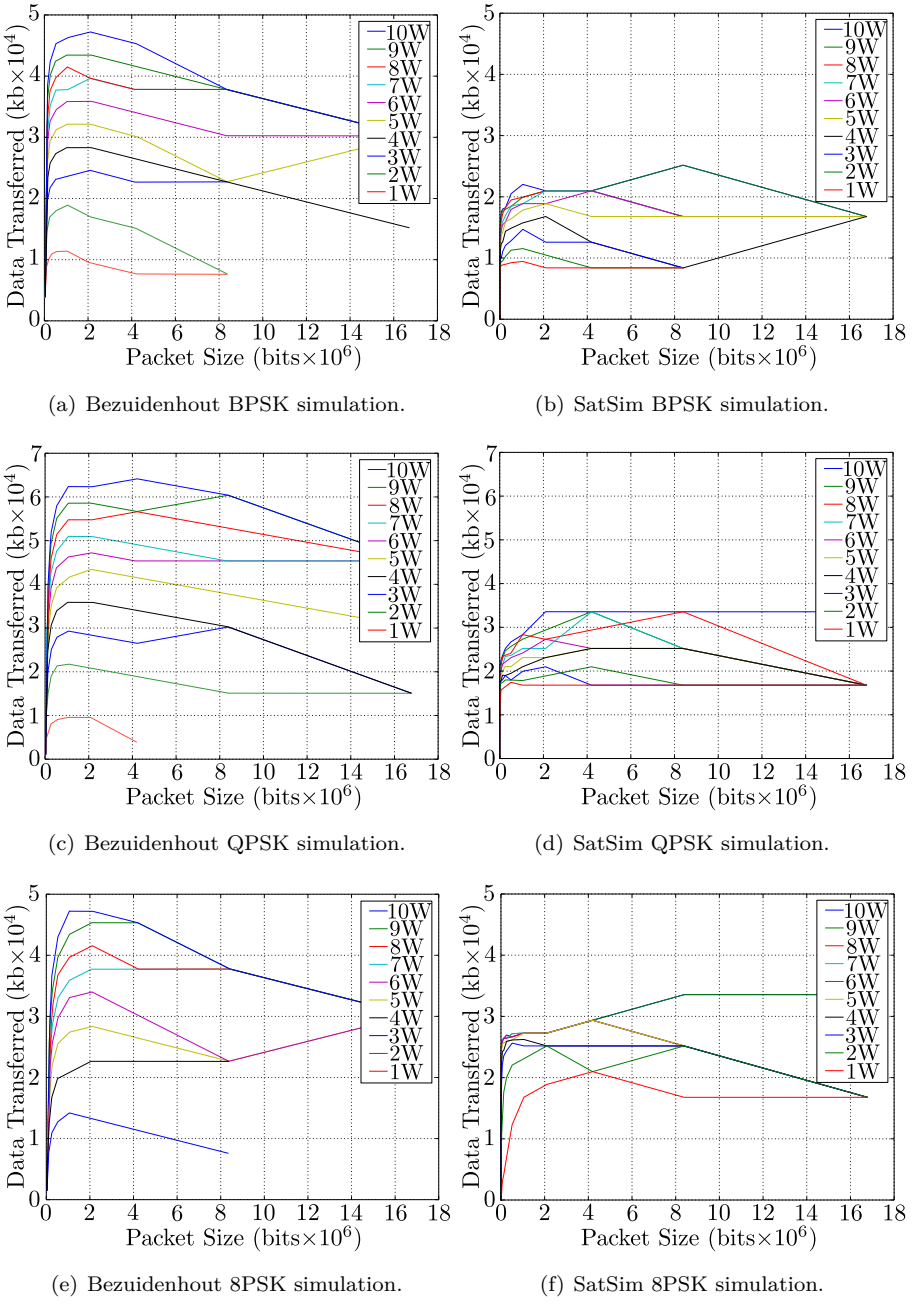


Figure 3.32 – A simulation with BPSK modulation and a modified antenna pattern with increased gains at lower elevation.

The full set of comparative results can be viewed in Figure 3.33. Another observation pointing towards inadequate antenna gain at low elevation, is the tight grouping of the higher transmission power values. The lack of antenna gain at the low elevations limits the gain of increased transmission power. When the antennas get closer to boresight, the links are within margin and then the same amount data transfer is observed. Figure 3.32 shows the results of a simulation with a modified antenna with increased gain at lower elevations. Vastly increased data transfer is observed and the higher transmission powers also show improved performance.



(a) Bezuidenhout BPSK simulation.

(b) SatSim BPSK simulation.

(c) Bezuidenhout QPSK simulation.

(d) SatSim QPSK simulation.

(e) Bezuidenhout 8PSK simulation.

(f) SatSim 8PSK simulation.

Figure 3.33 – Comparative simulation results to Bezuidenhout [8].

It has been established that the observed differences are most likely related to unclear documentation. SatSim is displaying similar behaviour and results after modifications were made in the test configuration. The SatSim simulations correlate with the other simulator's results and it is therefore safe to assume that SatSim can provide fairly accurate results.

3.5.4 Performance Analysis

Although discrete-event simulation provides a faster alternative to simulating a network, it cannot always guarantee fast execution times. The more events there are in a simulation, the longer the simulation will take to process. Early simulations illustrated this, as the execution times varied from a few seconds to tens of minutes, depending on simulation variables for the same simulated time.

As part of the verification tests, a timing test was conducted where the data packet sizes were varied and the simulations timed, to get an indication of performance. The transmission power was also varied between 1 W to 10 W, but this had little effect on the simulation time. The test script executed one simulation per processing core. Each simulation lasts for 900 simulated seconds and the processing time for each is measured. The results can be viewed in Figure 3.34.

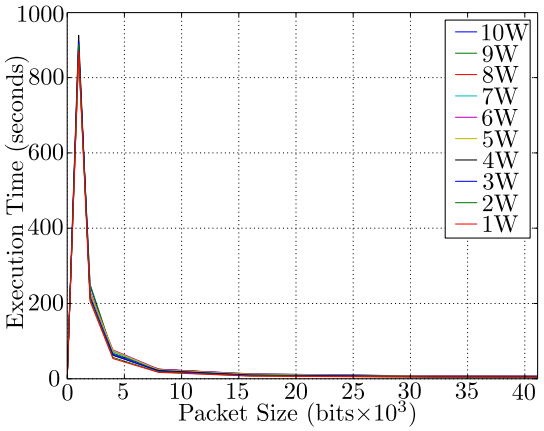


Figure 3.34 – Execution time compared to packet size for a 900 second simulation on an Intel Core i5-3570 processor with four processing cores clocked at 3.40 GHz.

From Figure 3.34 is clear that smaller packet sizes increase the execution time to 860 seconds, which is almost a 1-to-1 ratio with the simulated time. This is not acceptable performance and smaller increases in simulation time is expected, as the number of events does not increase drastically with a difference in packet sizes. The memory usage was another concern, as larger simulations could take up to 1 GB of memory. After a brief investigation, the problem was identified in the packet generation algorithm. The algorithm generated all scheduled packets at a single time. This is an issue, since all the generated packets will not always be transmitted before the end of the simulation, which leads to many redundant packets. A modification allowed for packets to be generated on demand. This decreased simulation times for small packet sizes drastically and reduced the amount of memory required for each simulation from around 150 MB to 15 MB. The improved simulation times can be viewed in Figure 3.35

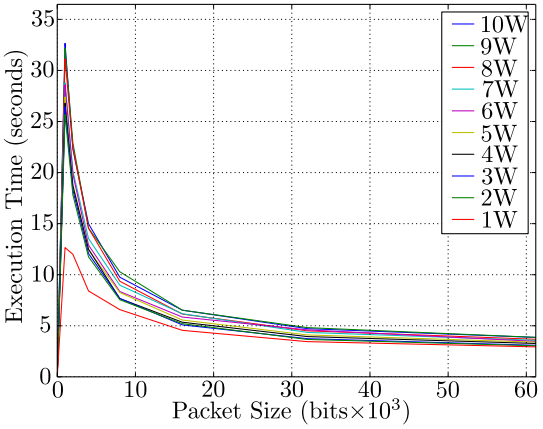


Figure 3.35 – Execution time compared to packet size for a 900 second optimised simulation on an Intel Core i5-3570 processor with four processing cores clocked at 3.40 GHz.

A final test was run to characterise the simulation performance. The test consisted of a simulation of 900 seconds with a data packet size of 32 kb and transmission power of 10 W, with the data transmission rate stepped from low to high in multiples of 76 kbps. The results can be viewed in Figure 3.36. The execution time shows a linear increase, which is to be expected, as the number of events in the simulator are generated at a constant rate. The execution time starts decreasing after 1064 kbps, since the link budget is getting too weak to transmit data packets, which in turn reduces the number of events in the simulation.

Simulation performance can be an issue, as it was with Bezuidenhout's work where a single simulation run could take up to 14 hours to complete [8]. The configuration for a 14 hour simulation was not documented and a direct performance comparison is unfortunately not possible. However, the performance of SatSim appears to be acceptable and should generate results in reasonable times, as the ratio between the execution time and simulated time is small. Care should be taken with the implementation of new protocols, as minor changes can have substantial performance implications.

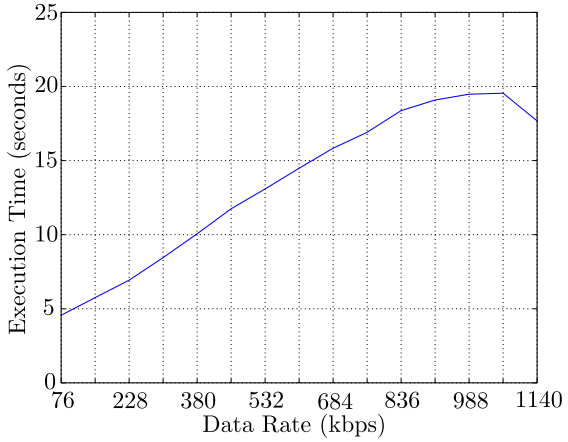


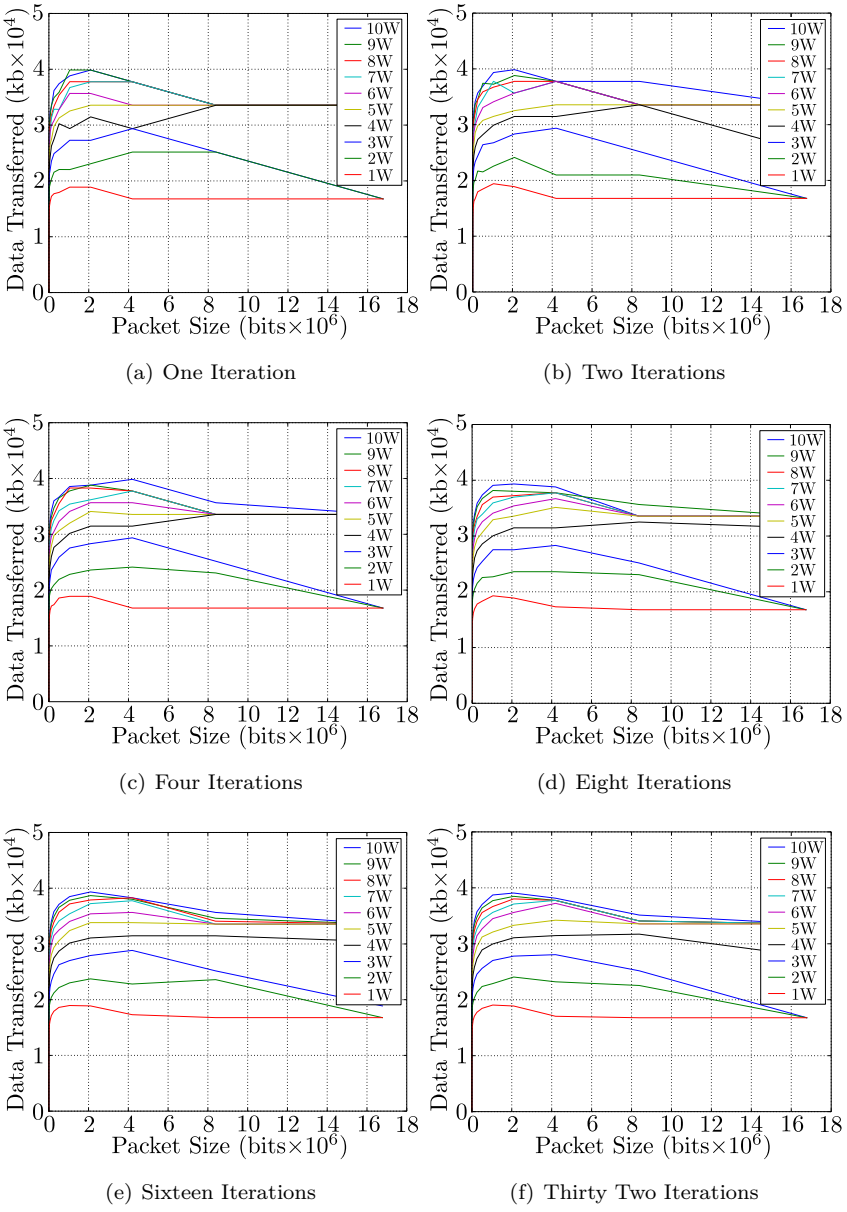
Figure 3.36 – Simulation with varying data transmission rates (multiples of 76 kbps) on an Intel Core i5-3570 processor with four processing cores clocked at 3.40 GHz.

3.5.5 Data Averaging

Due to the statistical nature of the simulation, specifically error generation, the results can vary between two consecutive simulations with the same configuration. This was observed with protocols that rely on RTS and CTS packets to establish a link. In one simulation the RTS and CTS packet will "sneak" through the channel without any errors and establish a link very early, where in the next simulation it can take minutes longer to establish the same link. This can, in some instances, yield vastly different results. A solution to this problem is to run a simulation multiple times and then take the average result over all the iterations. This is basically the application of the Monte Carlo method. It will provide a more accurate representation of the behaviour of the specific configuration.

This yields another question regarding how many iterations are required before the averaged behaviour will be reached and further iterations will not provide more accurate results. This type of simulation is time consuming and it is worth investigating what the optimal number of iterations would be. Figure 3.37 shows results that have been averaged over multiple iterations.

One iteration of the simulation results in a rough estimate of the trend of the simulation. The results appear to stabilise between eight and sixteen iterations. The difference between sixteen and thirty-two iterations is negligible when comparing the increase in execution time to the increase in statistical significance.



(a) One Iteration (b) Two Iterations

(c) Four Iterations (d) Eight Iterations

(e) Sixteen Iterations (f) Thirty Two Iterations

Figure 3.37 – Data averaging results.

3.6 Limitations

The simulator does have some limitations that the user must be aware of. Some of these limitations are inherent to discrete-event simulation and others can be addressed as part of future work.

- The earth is modelled as a perfect sphere.
- Communication relies only on LOS (signals do not diffract).

- The link budget does not account for atmospheric attenuation, as it is minimal for lower frequency signals.
- The RF channel does not simulate burst errors. Only random bit errors due to channel noise is currently simulated.
- There is no weather model present. This is discussed in Section 3.4.1.3.
- Frame-based simulation leads to the calculation of a single BER value for an entire frame. This can yield inaccurate results, if a frame takes a long time to transmit and the changes in geometry occur relatively fast. However, this is a reasonable assumption if frame sizes are not too large, compared to the data rate.
- The are no channel access models. It is not possible to simulate multiple transmitting nodes accurately, as frame collisions are not currently simulated. This is not a requirement for this project, as only single links will be investigated.
- Doppler effect is not simulated. This will have little effect on most simulations, but it can cause problems depending on orbits. Currently, it is assumed that all receivers and transmitters can compensate for Doppler shifts.
- Satellites can currently only orbit in circular orbit paths.

Chapter 4

Network Protocol Investigation

With an adequate simulator developed in the form of SatSim, it is now possible to investigate an optimal network protocol strategy for nano-satellite communications.

The investigation will take a look at relevant, existing protocols that are used within the nano-satellite industry and will attempt to improve and optimise them. A similar approach to the CubeSat development methodology will be followed, which is to use existing technologies as far as possible. By using existing technologies in the network protocol design, future users will have a frame of reference for implementing the proposed improvements or implementations.

The first protocol that was identified for analysis was AX.25 (discussed in Section 4.2). It was established in Section 1.4 that AX.25 is one of the most common protocols implemented in nano-satellite missions which makes it a logical choice to use as a base in this investigation. FX.25 is the other existing protocol chosen for analysis (discussed in Section 4.3). FX.25 is an attempt to improve on AX.25, by providing additional FEC capabilities, while maintaining backward compatibility with AX.25.

4.1 Network Protocol Basics

This section briefly discusses some groundwork to assist with understanding and analysing network protocols.

4.1.1 Packet Radio

Packet radio is the boundary where digital and analogue communications meet. Packet radio is similar to a telephone network, where the telephone modem is replaced by a Terminal Node Controller (TNC), the telephone is replaced by a radio transceiver, and the phone system is replaced by radio waves [9]. Packet radio can take any form of data stream and transmit it via radio waves to another station. Figure 4.1 illustrates the basic layout of a packet radio station. This setup is scaled down with appropriate hardware for deployment on CubeSats.

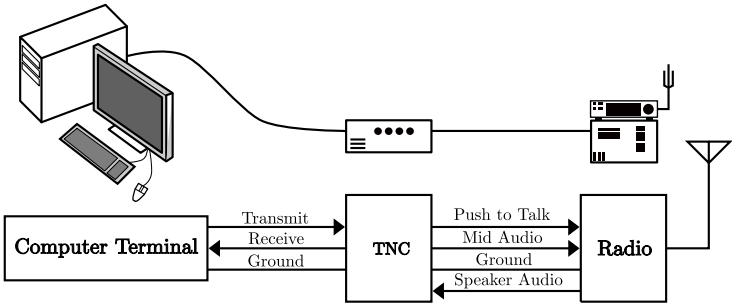


Figure 4.1 – Example of a packet radio station. Adapted from Ref. [9].

The computer acts as the data terminal, which passes and receives information to and from the TNC. The TNC contains a modem and some form of microprocessor to convert the data received from the computer into the chosen packet format. The TNC constructs that packet and applies any additional functionality, such as FEC or check sums. The TNC then modulates the data into an audio signal, which will then be broadcasted over the attached radio. This process is reversed when data is being received via the radio.

4.1.2 OSI Model

The Open Systems Interconnection (OSI) model is a seven layer model [51] defined by the International Organisation for Standardisation (ISO). This model assists with classifying protocols into a logical hierarchy, according to the defined functionality. The OSI model can be viewed in Figure 4.2. The lines between some layers are often blurred, for example the data link and network layer are in some cases tightly coupled, making it difficult to distinguish certain layers.

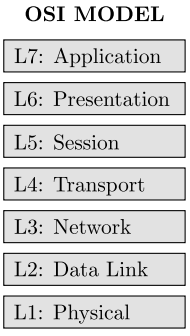


Figure 4.2 – The seven layer OSI model.

The physical layer (L1) defines how information is physically transmitted over a channel. This includes electrical characteristics and the type of transfer medium. The data link layer (L2) provides a link between two nodes in a network. This layer encodes bits into frames and defines how the physical medium is accessed. The network layer (L3) defines

how information is passed between a network of nodes. This includes routing and how a certain data fragment is passed via multiple nodes to its destination. The transport layer (L4) provides the procedures for data transmission through multiple networks. The session layer (L5) controls connections between devices on the network. It manages connectivity by opening and closing connections as required. The presentation layer (L6) defines how data is presented to the application layer. It translates between network and application data. The application layer (L7) interacts directly with the software application.

4.1.3 Framing

Framing is a method used to structure data in such a way that it can be recovered and recognised by a receiver from a sequence of bits. The primary objective of framing is to identify the frame boundaries and the fields within the frame. Frame synchronisation (mostly referred to as framing) is the process of defining and identifying these frame boundaries. Frame boundaries are indicated by flags (also known as delimiters), which can be a specific bit or byte sequence.

A frame consists of many different fields, for example the flags/delimiters, a control field, a data field and a frame check sequence field. Most radio protocols are based on the High-Level Data Link Control (HDLC) definition. HDLC is a bit-orientated protocol, but byte oriented protocols also exist. These protocols rely on the fields within a frame to consists of an integer number of bytes, which is the case, for example, with the Point-to-Point Protocol (PPP). Byte orientated protocols perform byte stuffing to achieve transparency. This usually involves the insertion of an escape sequence to prevent erroneously detecting a frame delimiter.

Framing is an important aspect of network protocols, especially with RF networks, where bit errors can influence the escape sequences and delimiters.

4.2 AX.25

This section covers the AX.25 protocol (version 2.2) and its implementation in SatSim. It is a commonly used protocol by amateur radio enthusiasts and nano-satellite developers. A brief description of AX.25 is provided, as it brings into perspective the complexity involved in network protocols. The complete AX.25 definition can be viewed in Ref. [10].

4.2.1 AX.25 Protocol Definition

There are three categories of AX.25 frames, which are inherited from HDLC: an information frame (I frame), a supervisory frame (S frame) and an unnumbered frame (U frame). Each frame is composed of fields to carry information. The AX.25 frames and their respective fields can be viewed in Figure 4.3.

Flag	Address	Control	Info	FCS	Flag
01111110	112/224 bits	8/16 bits	N*8 bits	16 bits	01111110

(a) AX.25 U and S frame construction. Adapted from Ref. [10].

Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/224 bits	8/16 bits	8 bits	N*8 bits	16 bits	01111110

(b) AX.25 I frame construction. Adapted from Ref. [10].

Figure 4.3 – The AX.25 frame constructions. The Info field only exists in certain frames and all fields contain an integer number of bytes.

The full AX.25 specification defines acknowledgements of frames carrying information. The protocol tracks successfully received frames with sequence numbers. It operates on a rolling window, where frames up to the end of the window can be acknowledged in one frame. This is done to reduce protocol overhead.

AX.25 makes use of a Cyclical Redundancy Check (CRC) to determine the integrity of data in a frame. The CRC value is named the Frame Check Sequence (FCS) in the protocol definition. If a CRC error is detected by the receiver, it will respond with a REJECT frame (see Section 4.2.1.3). This frame acknowledges all the previously received information frames and then requests the re-transmission of the corrupted frame. If no frames are corrupted, the receiver will respond with a Receiver Ready (RR) frame to acknowledge the successful frames.

If the transmitter does not receive any response from the receiver (REJECT or RR frame) it will time-out and send a RR frame in an attempt to poll the receiver. The receiver will then respond with either a Receiver Not Ready (RNR), REJECT or RR frame. The RR frame will continue transmission from the last transmitted frame. The REJECT response will resume transmission from where the frame sequence was corrupted. The RNR response indicates that the receiver is currently not able to receive information frames. This can be due to a full receive buffer. The transmitter will wait a certain amount of time before polling

the receiver again, in the event of a RNR frame. This operational sequence allows AX.25 to re-transmit frames that were lost or corrupted during transmission.

4.2.1.1 Flag Field

The flag field delimits a frame and occurs at the start and end of each frame. The pattern is one byte in length, and is based on the HDLC specification. The flag field's binary sequence is '0111110' (0x7E). As this pattern is now reserved as a flag, it may not be repeated anywhere else inside the frame. To achieve transparency and prevent the bit sequence from repeating in the frame it is necessary to implement bit stuffing. If the transmitter detects five consecutive ones, it automatically adds an additional zero before continuing. If the receiver detects five consecutive one bits, it will check for the following conditions to decode the message:

- If the next bit is a '0', then it will be removed to obtain the original sequence.
- If the next two bits are '10', then the flag is detected.
- If the next two bits are '11', then the frame contains errors.

Bit stuffing is done on all fields, except the flag field. Two consecutive frames can share a flag, where the end flag of one frame will be the start flag of the next frame. This is more efficient in situations where large amounts of data is transmitted consecutively.

4.2.1.2 Address Field

The address field contains the source and destination address of the specific frame as well as an optional two Layer 2 repeater sub-fields. Frame repeating is considered a higher level protocol function and is slowly being phased out from AX.25, but support for two repeaters remain for backward compatibility. This discussion will consider only non-repeater operational mode and thus the address field encoding will not make use of repeaters.

Address Field	
Destination Address Subfield	Source Address Subfield
A1 A2 A3 A4 A5 A6 A7	A8 A9 A10 A11 A12 A13 A14

A1	Call Sign	A8	Call Sign
A2	Call Sign	A9	Call Sign
A3	Call Sign	A10	Call Sign
A4	Call Sign	A11	Call Sign
A5	Call Sign	A12	Call Sign
A6	Call Sign	A13	Call Sign
A7	CRRSSID0	A14	CRRSSID1

Figure 4.4 – Address field encoding for non-repeater mode. The LSB of each byte is the HDLC extension bit. It is set to zero on all but the last byte in the address field. The 'R' bits are reserved and used in an agreed-upon manner in individual networks. The 'C' bit is the command/response bit of an AX.25 frame. Adapted from Ref. [10].

Figure 4.4 shows the address field encoding. Each field consists of a call-sign and a Secondary Station Identifier (SSID). The call-sign consists of 6 upper-case alpha numerical ASCII characters and the SSID is a 4-bit integer that differentiates between stations using the same call-sign. The address field is extended beyond one byte by setting the Least Significant Bit (LSB) of each byte (the extension bit) to zero. A zero indicates that the next byte contains more addressing information and a one indicates the end of the addressing information. This process involves shifting each bit of the 6 ASCII call sign characters to the left and assigning the LSB to a one or zero.

The SSID byte at the end of each address sub-field (byte A7 and A14) contains the SSID and the 'C' bit (the 7^{th} bit). The C bit identifies which version of the protocol is implemented. If both C bits are set to zero, then the distant device is using the older version of the protocol. The new version always has one of the two bits set to one and the other to zero, depending on whether the frame is a command or response. The encoding of the command/response information can be viewed in Figure 4.5 and an example of an I frame can be viewed in Table C.1.

Frame Type	SSID C-Bit	Source SSID C-Bit
Previous Versions	0	0
Command (V.2.X)	1	0
Response (V.2.X)	0	1
Previous Versions	1	1

Figure 4.5 – Command/Response encoding. Adapted from Ref. [10].

4.2.1.3 Control Field

The control field identifies which type of frame is being sent or received. As mentioned before, the three types of frames are the I, U and S frames. The control field can be 1 or 2 bytes long and may use sequence numbers to maintain link integrity. The sequence numbers can be a 3-bit (modulo 8) or 7-bit (modulo 128) integer.

Figure 4.6 shows the structure of the control field. N(S) is the send sequence number. N(R) is the receive sequence number. The 'S' bits are supervisory function bits. The 'M' bits are the unnumbered frame modifier bits. The P/F bit is the Poll/Final bit.

The default mode is modulo 8. In this mode the sequence number is assigned a value from 0 to 7, which allows up to 7 outstanding frames per connection. Modulo 128 operation allows up to 127 outstanding frames per connection.

I Frame Control Field: The I frame control field contains the transmitter and receiver sequence numbers only, as shown in Figure 4.6. There are no additional sub-fields for an I frame.

Frame Type	Control Field Bits							
	7	6	5	4	3	2	1	0
I Frame	N(R)			P	N(S)			0
S Frame	N(R)			P/F	S	S	0	1
U Frame	M	M	M	P/F	M	M	1	1

(a) Modulo 8 sequence number (3-bit) frame.

Frame Type	Control Field Bits															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I Frame	N(R)							P	N(S)							0
S Frame	N(R)							P/F	0	0	0	0	S	S	0	1

(b) Modulo 128 sequence number (7-bit) frame.

Figure 4.6 – AX.25 control field formats.

S Frame Control Field: The possible field configurations for the S frame control field can be viewed in Figure 4.7.

Frame Type	Control Field Bits							
	7	6	5	4	3	2	1	0
Receive Read (RR)	N(R)			P/F	0	0	0	1
Receive Not Ready (RNR)	N(R)			P/F	0	1	0	1
Reject (REJ)	N(R)			P/F	1	0	0	1
Selective Reject (SREJ)	N(R)			P/F	1	1	0	1

(a) Modulo 8 S frame control field.

Frame Type	Control Field Bits															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Receive Read (RR)	N(R)							P/F	0	0	0	0	0	0	0	1
Receive Not Ready (RNR)	N(R)							P/F	0	0	0	0	0	1	0	1
Reject (REJ)	N(R)							P/F	0	0	0	0	1	0	0	1
Selective Reject (SREJ)	N(R)							P/F	0	0	0	0	1	1	0	1

(b) Modulo 128 S frame control field.

Figure 4.7 – AX.25 S frame control field. Adapted from Ref. [10].

An S frame can deliver one of the following commands:

- **Receiver Ready (RR)** - Indicates that the system is ready to receive more I frames. It also acknowledges all I frames up to and including N(R)-1 and clears a busy condition that was previously set by a RNR command. The status of a receiver can be requested by sending an RR command, with the P-bit set to one.
- **Receiver Not Ready (RNR)** - Indicates that the receiver is currently busy and cannot accept more I frames. It also acknowledges I frames up to N(R)-1. Frame N(R) and above, that may have been transmitted, is discarded and must be re-transmitted when the busy condition clears. The condition can be cleared by sending a UA, RR,

REJ or SABM(E) frame. The status of a receiver can be requested by sending an RNR command, with the P-bit set to one.

- **Reject (REJ)** - This frame requests the re-transmission of I frames starting with N(R). All frames with sequence numbers N(R)-1 or less are acknowledged. Additional I frames can be appended to the re-transmission of the N(R) frame. Only a single rejection condition is allowed in each direction at a time. The condition is cleared by the correct reception of the I frame that was originally rejected. The status of an receiver can be requested by sending a REJ command, with the P-bit set to one.
- **Selective Reject (SREJ)** - This frame is used to request the re-transmission of the N(R) frame. If the P/F bit is set to one, the I frames up to N(R)-1 are acknowledged. If the P/F bit is set to zero, the I frames are not acknowledged. The SREJ condition is cleared when the N(R) frame is successfully received (where N(R) equals N(S)).

U Frame Control Field: The U frame control field is either command frames or response frames and their descriptions are extensive. The available frame types can be viewed in Figure 4.8. The effects of these commands can be studied in Ref. [10] (pages 22 to 30). Of specific importance is the XID frame, which is used to negotiate parameters for a specific connection.

Frame Type	Type	Control Field Bits							
		7	6	5	4	3	2	1	0
Set Async Balanced Mode (SABME)	Cmd	0	1	1	P	1	1	1	1
Set Async Balanced Mode (SABM)	Cmd	0	0	1	P	1	1	1	1
Disconnect (DISC)	Cmd	0	1	0	P	0	0	1	1
Disconnect Mode (DM)	Res	0	0	0	F	1	1	1	1
Unnumbered Acknowledge (UA)	Res	0	1	1	F	0	0	1	1
Frame Reject (FRMR)	Res	1	0	0	F	0	1	1	1
Unnumbered Information (UI)	Either	0	0	0	P/F	0	0	1	1
Exchange Identification (XID)	Either	1	0	1	P/F	1	1	1	1
Test (TEST)	Either	1	1	1	P/F	0	0	1	1

Figure 4.8 – AX.25 U frame control field. Adapted from Ref. [10].

4.2.1.4 PID Field

The Protocol Identifier Field (PID) identifies what protocol, if any, is running on top of the AX.25 layer. The list of possible protocols can be viewed in Table C.2.

4.2.1.5 Info Field

The info field carries user data. This field defaults to 256 bytes. It is only present in the I, UI, XID, TEST and FRMR frames.

4.2.1.6 FCS Field

The FCS is a 16-bit value, that is calculated by the transmitter and receiver, to determine if a frame has been corrupted during transmission. The FCS is calculated as specified in the HDLC reference document (ISO 3309). This check sequence is a 16-bit CRC-CCITT calculation, with a generator polynomial of $G(x) = x^{16} + x^{12} + x^5 + 1$ (or 0x1021). It is calculated over all the fields, excluding the delimiting flags.

4.2.1.7 Bit Transmission Order

The FCS field is transmitted with the Most Significant Bit (MSB) of each byte sent first. All other fields are transmitted with the Least Significant Bit (LSB) of each byte first.

4.2.1.8 Invalid Frames

A frame is considered invalid in the following scenarios:

- Consists of less than 136 bits (including the delimiting flags).
- Is not bounded by opening and closing flags.
- Does not contain an integer number of bytes.

4.2.2 AX.25 Simulation Implementation

AX.25 is a relatively comprehensive protocol and is rarely fully implemented on all nano-satellite missions. The low orbit altitudes lead to short ground station access times and the full AX.25 stack requires some overhead to establish a link. A sub-set of AX.25 is often implemented on nano-satellite missions to reduce complexity and overhead. This sub-set is known as 'connectionless AX.25'. In this mode, the AX.25 protocol operates in a connectionless manner, where data can be exchanged between nodes without any prior arrangement or configuration of a network link.

Connectionless AX.25 makes use of only a single type of frame, which is the Unnumbered Information Frame (UI Frame) as in Figure 4.9.

Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/224 bits	00000011	11110000	N*8 bits	16 bits	01111110

Figure 4.9 – Connectionless AX.25 UI Frame.

The control field is set to 0x03 to indicate that the frame is a UI Frame (see Figure 4.8). The PID field is set to 0xF0 to indicate that there is no layer 3 protocol (see Table C.2) The address and FCS fields are encoded as discussed in Section 4.2.

AX.25 in connectionless mode is basically only a framing protocol to transfer data between two or more nodes. It does not guarantee reception and does not re-transmit any lost frames. If any of these are required, it will be implemented on top of the AX.25 layers.

The connectionless AX.25 protocol was implemented within the SatSim framework as it is mostly used within the nano-satellite environment. Connectionless AX.25 allows for a slightly simpler implementation and can form the foundation of a full AX.25 implementation, as a full implementation is currently not required.

The AX.25 protocol implementation is configured to simulate down to bit-level. This is required as the protocol makes use of bit-level operations such as bit-stuffing and the detection of delimiting flags.

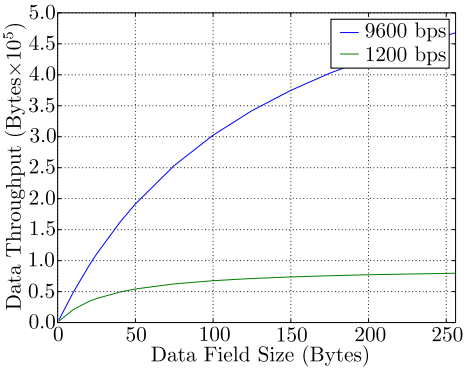
4.2.3 AX.25 Simulation Verification

To test if the implementation yields realistic results, it needs to be compared to previous results. Not a single previously published AX.25 throughput simulation could be found in the available literature. Grønstad [52] documents the implementation of AX.25 for the CubeSTAR project. A theoretical throughput calculation was done in the document to determine throughput estimates. The theoretical calculations included measured processing delays on the hardware platform. The configuration for the theoretical throughput was:

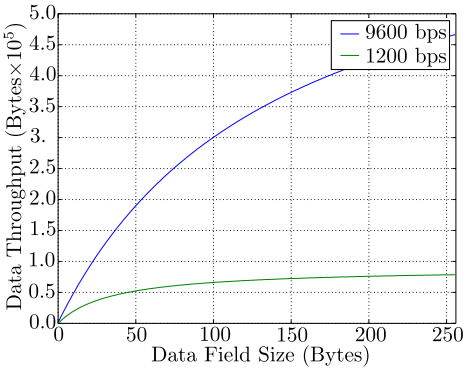
- Constant propagation distance of 2500 km.
- A 600 second satellite pass.
- The BER is independent and uniformly distributed .
- Calculated at 1200 and 9600 bps bit rates.
- Calculated over a range of information field sizes from 0 to 256 bytes.
- Calculated for BERs of 0, 10^{-5} and 10^{-4}.

The simulator was configured with a constant BER RF channel and conformed to all the above listed items, except for the propagation distance, which changes as the satellite passes over the ground station. This has little to no influence on the simulation, as the BER is constant and will not vary with propagation distance. The AX.25 implementation also received an additional 100 millisecond delay between frames as part of the processing delay.

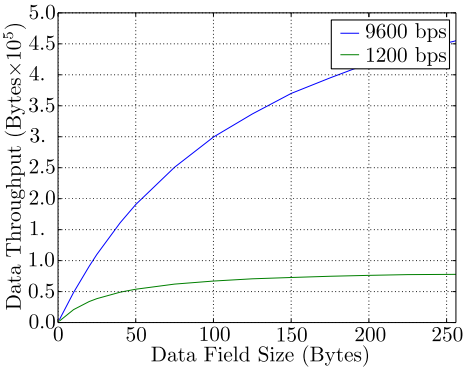
The simulations were executed and compared to the results of Grønstad and can be viewed in Figure 4.10. The results are nearly identical. Some variance was expected, as the simulation implementation filled the information field with random data and this varied the number of additional bits added by the bit stuffing procedure. A theoretical sanity check showed that a total of 1827 frames can be transmitted in the 900 second pass with a bit rate of 9600. The simulation yielded 1832 frames in the pass, which is within range of the theoretical expectation.



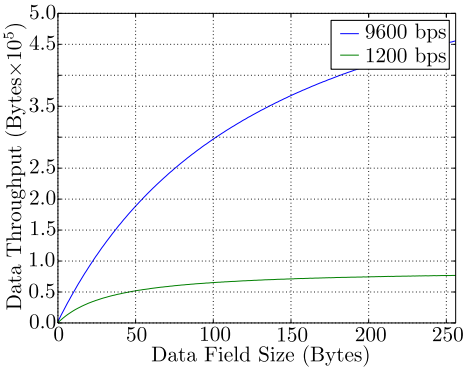
(a) Simulated connectionless AX.25 with a constant BER of 0.



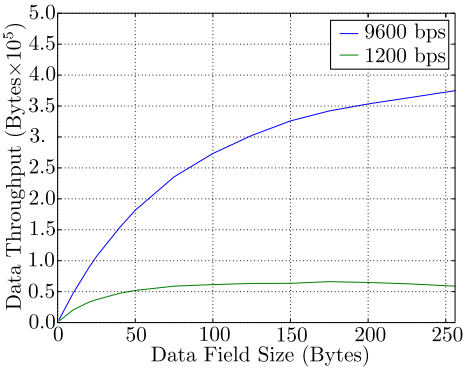
(b) Theoretical connectionless AX.25 with a constant BER of 0. Adapted from Ref. [52].



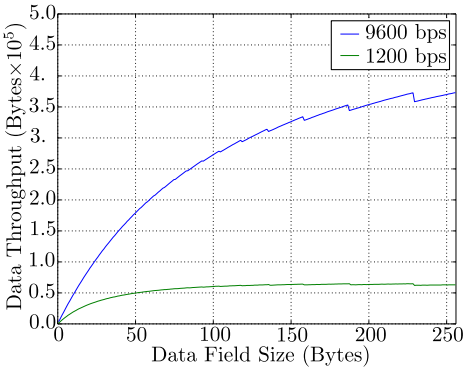
(c) Simulated connectionless AX.25 with a constant BER of 10^{-5}.



(d) Theoretical connectionless AX.25 with a constant BER of 10^{-5}. Adapted from Ref. [52].



(e) Simulated connectionless AX.25 with a constant BER of 10^{-4}.



(f) Theoretical connectionless AX.25 with a constant BER of 10^{-4}. Adapted from Ref. [52].

Figure 4.10 – Theoretical and simulation throughput of various BER channels over a pass of 600 seconds.

4.3 FX.25

AX.25 has the drawback that a single bit error can cause the loss of a frame, sometimes even two, due to the lack of FEC. The FX.25 [11] wrapper was developed by the Stensat group in 2005 to assist with this problem by wrapping the AX.25 frame with added FEC data. The wrapping is done in such a way that backward compatibility with AX.25 is maintained. A station without the ability to decode FX.25 frames will still be able to decode the embedded AX.25 frame, as the preamble and postamble will be discarded by normal AX.25 operating procedures.

4.3.1 FX.25 Protocol Definition

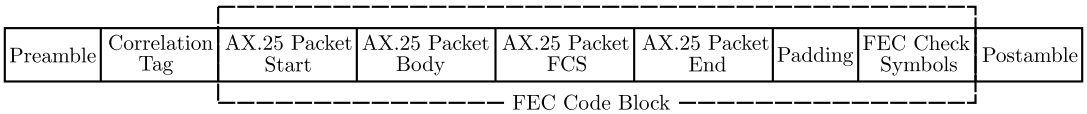


Figure 4.11 – FX.25 Structure. Adapted from Ref. [11].

Figure 4.11 illustrates the wrapper for FX.25. The preamble block is the sequence 0x7E, which is the same as AX.25's delimiter flag, which allows the receiver to acquire the signal. This involves the receiver breaking through squelch, performing Automatic Gain Control (AGC) and also performing clock recovery. A recommended minimum of 4 bytes should be transmitted, which is the sequence 0x7E7E7E7E. Additional bytes may be required, depending on the expected receiver characteristics and the transmitted line rate, but the protocol description does not expand further on this topic. Preamble bytes can be used for determining byte alignment, when used in conjunction with the correlation tag.

The correlation tag is a 64-bit sequence, designed to indicate the start of the frame and identify which FEC algorithm is applied. The tag is located outside the FEC code block as it is designed to provide good correlation even in the presence of channel errors. This allows the receiver to detect the start of the frame even when there are errors within the 64-bit correlation tag. The tags are generated with Gold Codes, which provide high auto-correlation and low cross-correlation. These codes are simple to generate, which makes it ideal for this application. More detail regarding the code generation is available in FX.25 protocol definition [11]. Reed-Solomon (RS) FEC coding is used in FX.25, as it has been used successfully in other terrestrial and space radio applications. The various tags and their assignments can be viewed in Table 4.1

The FEC code block is the data over which the FEC algorithm is applied. The size of the FEC code block is dependent on the specific FEC scheme implementation. The padding field is required to pad out the code block to the appropriate number of bits for the FEC scheme. The size of the padding field will be different for every frame. The padding field does not carry any data and will be discarded after reception.

Table 4.1 – FEC algorithm and correlation tag value assignment. Adapted from Ref. [11].

Tag	Correlation Tag Value	FEC Coding Algorithm
Tag_00	0x566ED2717946107E	Reserved
Tag_01	0xB74DB7DF8A532F3E	RS(255,239) 16-byte check value, 239 information bytes
Tag_02	0x26FF60A600CC8FDE	RS(144,128) - shortened RS(255,239), 128 info bytes)
Tag_03	0xC7DC0508F3D9B09E	RS(80, 64) - shortened RS(255,239), 64 info bytes
Tag_04	0x8F056EB4369660EE	RS(48, 32) - shortened RS(255,239), 32 info bytes
Tag_05	0x6E260B1AC5835FAE	RS(255,223) 32-byte check value, 223 information bytes
Tag_06	0xFF94DC634F1CFF4E	RS(160,128) - shortened RS(255,223), 128 info bytes
Tag_07	0x1EB7B9CDBC09C00E	RS(96, 64) - shortened RS(255,223), 64 info bytes
Tag_08	0xDBF869BD2DBB1776	RS(64, 32) - shortened RS(255,223), 32 info bytes
Tag_09	0x3ADB0C13DEAE2836	RS(255,191) 64-byte check value, 191 information bytes
Tag_0A	0xAB69DB6A543188D6	RS(192,128) - shortened RS(255,191), 128 info bytes
Tag_0B	0x4A4ABEC4A724B796	RS(128, 64) - shortened RS(255,191), 64 info bytes
Tag_0C ↓ Tag_3F		Undefined
Tag_40	0x41C246CB5DE62A7E	Reserved

The specification recommends the insertion of the 0x7E sequence for the padding field, as this is identified as null frames by AX.25. In the event of a non-integral number of bytes being present in a frame, a fraction of the 0x7E sequence will be inserted in the byte that forms part of the closing flag of the AX.25 frame. The LSB position of the flag will be truncated to fill the byte.

The postamble is a series of 0x7E bytes intended to provide separation between the end of the FX.25 frame and the transmitter un-key event. The specification recommends that a minimum of 2 bytes be transmitted which is a single sequence of 0x7E7E. Additional bytes may be necessary, depending on the receiver characteristics and the transmitted line rate.

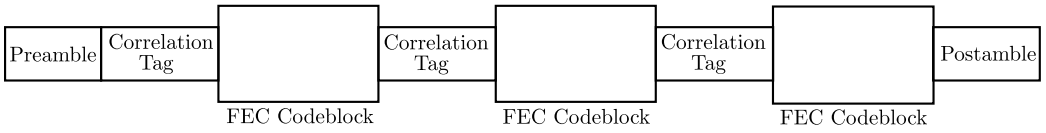


Figure 4.12 – FX.25 Streaming. Adapted from Ref. [11].

It is worth noting that the AX.25 bit-stuffing algorithm is not safe to use outside the application of AX.25. If it were implemented on a FX.25 frame, it could potentially nullify the FEC capabilities when a single bit error occurs. For the FEC algorithm to function correctly, it requires that the FEC check symbols are in the correct locations in the frame. If a bit error occurs and the bit stuffing algorithm adds or removes a single bit, it will invalidate the FEC correction capabilities.

The FX.25 specification allows for the streaming of data as well. This involves reducing overhead by reducing the occurrences of the preamble and postamble as in Figure 4.12. The data is streamed with a single preamble and postamble - with a correlation tag separating each code block.

4.3.2 FX.25 Simulation Implementation

A challenging aspect of FX.25 is the RS FEC. Encoding and decoding of FEC on data is beyond the scope of this project, as it is a highly complex process. It also requires a lot of processing time, especially for decoding. The FEC encoding behaviour will thus be simulated as closely as possible to decrease complexity and increase processing performance.

The RS algorithm divides the data into symbols with 1 byte per symbol [53]. The code is able to fix one bit error per symbol, or a complete byte, when all the bits are erroneous in a byte. For example RS(255,223) can correct 16 errors, if each error is in a different byte, or if 16 complete byte errors occurs, it will correct all 16 bytes (16×8 bits). This characteristic makes RS coding effective for channels with burst errors. The default number of preamble and postamble bytes was set to 16. The number of preamble and postamble bytes have a small effect on the throughput, as streaming capability described in Figure 4.12 reduces the occurrences of the preamble and postamble bytes.

With the first implementation of FX.25, it was assumed that the preamble and postamble were used to delimit the frames similar to the implementation in the AX.25 protocol. This was quickly proven incorrect, as the preamble and postamble was easily corrupted and the frame made undetectable. This form of frame detection removes the advantages of having the FEC in the frame. After some further investigation it was discovered that the preamble was only used for the radio to lock onto the signal. The implementation was adapted to make use of a rolling window to detect the correlation tag. The first 64 bits of the incoming bit stream are used to do a correlation calculation on all the possible tags. If a valid tag is not identified, it removes one byte from the buffer and then repeats the correlation check. When a valid tag is detected, the block size of the FEC algorithm can be identified and removed for further decoding.

This implementation was similar to the working example provided by the FX.25 developers [54]. The working example also used a rolling window implementation on the bit stream. This is a computationally expensive solution, but it is the only reliable way of detecting the correlation tags. One assumption in the simulation is that the preamble is long enough for the receiver to detect and lock onto the signal in noisy channels.

Since the correlation tag can sustain bit errors during transmission, it will not always correlate fully with an available tag. A minimum threshold correlation value was set, to help determine if a correlation tag can be assumed correct. A threshold value of 0.9 was chosen after initial testing.

At this point it would have been desirable to compare the protocol implementation results to other academic papers to verify results, but unfortunately no sources with documented FX.25 results could be found (AX.25 sources are also very limited).

The potential improvements of FX.25 over AX.25 is unknown at this point. Both protocols will be simulated concurrently in SatSim. Not only will this provide an idea of the performance of FX.25, but it also serves as a verification of the FX.25 implementation. FX.25 is expected to provide less throughput than AX.25, but should be more reliable, especially over noisy channels.

4.4 AX.25 vs FX.25

An important merit that could be used to compare AX.25 to FX.25, is how the throughput of payload data is affected with changes in various parameters. Comparative simulations were done using the following configurations:

- Two ground transmitters were used: one sent AX.25 frames and the other FX.25 frames on different frequencies (437.3 MHz and 437.5 MHz), at 9600 bps with BFSK modulation.
- Two satellite receivers were used: one receiving AX.25 and the other receiving FX.25, in a 600 km circular orbit, passing precisely over the ground transmitters.
- The ground station's transmitting antennas are similar to the antenna in Figure 3.27(a).
- The satellite receiving antennas are 0 dB, omni-directional antennas.
- The FX.25 protocol uses a symbol size of 8 bits with a 16-byte (worst-case) check value.
- The duration of a single satellite pass is roughly 600 seconds.
- The throughput measurement is the actual amount of payload data that is received by the satellites.
- The data field sizes of the protocols are varied with each simulation.

The first batch of simulations were generated over various constant BER channels. All simulations were done without any processing delays, which equates to constant streaming of data from the ground station. The first simulation was with a 0 BER channel, as this provides a baseline to compare all other results to. The simulation results for a 0 BER channel can be viewed in Figure 4.13.

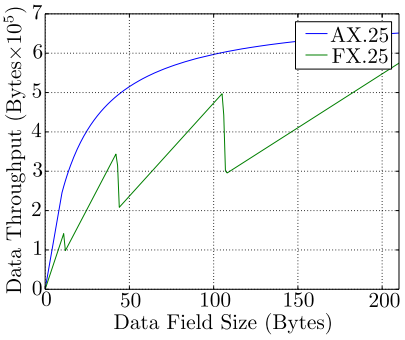
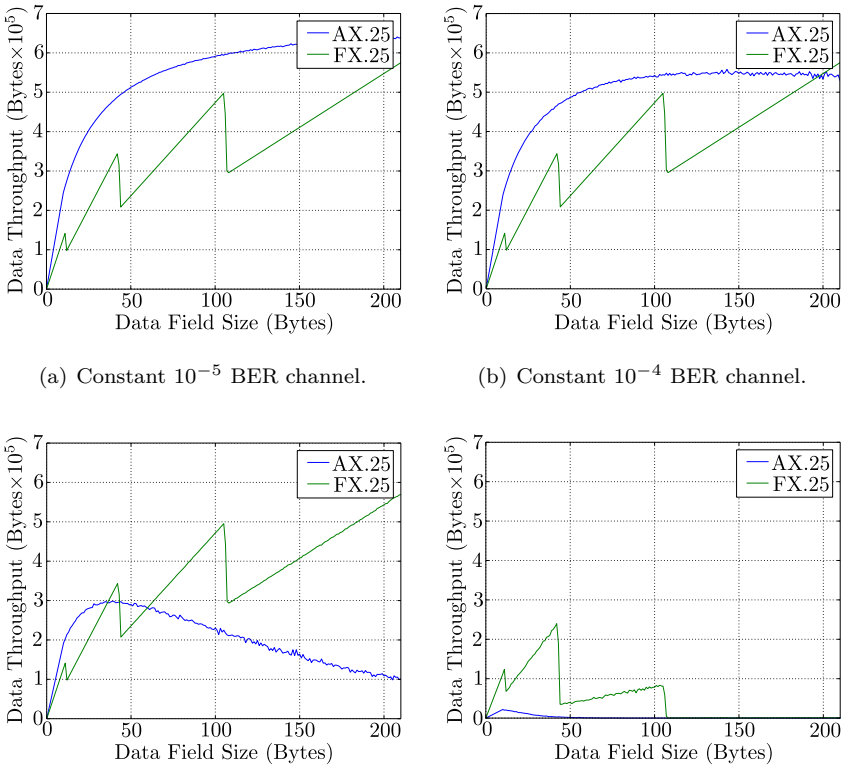


Figure 4.13 – Payload throughput for a 0 BER channel. The Data field size is limited by the maximum code block size of FX.25.

The saw-tooth pattern on the FX.25 throughput is caused by the automated choice of FEC. The FEC requires a set block size for the algorithm to work and pads the data to fill up the block if required. As the size of the the AX.25 frames increases with the increased data field size, the AX.25 frame size will approach the information block size limits of the FEC and will move to a larger block size. At this point the payload throughput drops dramatically, as the frame contains plenty of redundant padding data.



(a) Constant 10^{-5} BER channel.

(b) Constant 10^{-4} BER channel.

(c) Constant 10^{-3} BER channel.

(d) Constant 10^{-2} BER channel.

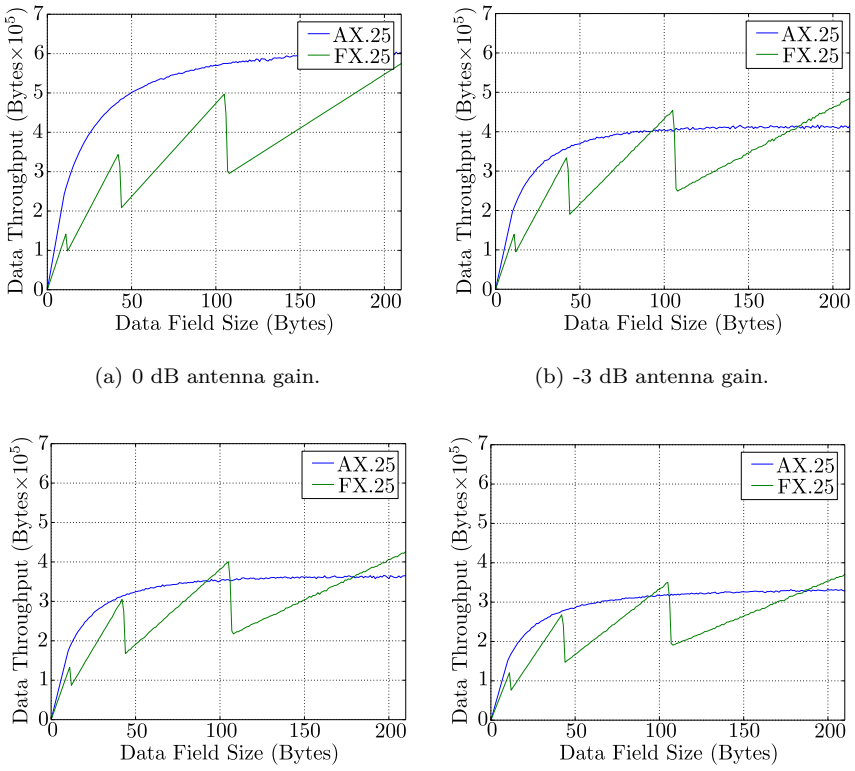
Figure 4.14 – Throughput of various BER channels.

The throughput for FX.25 is considerably smaller than that of AX.25 with a 0 BER channel, but this is to be expected. FX.25 transmits additional data for the FEC which is redundant for a 0 BER channel. The throughput should change in favour of FX.25 when a certain amount of bit errors are introduced into the channel. Simulation on various constant bit rate channels can be viewed in Figure 4.14.

From Figure 4.14 it is obvious that AX.25 has greater throughput up to a BER of 10^{-4}. This is to be expected, as bit errors are limited. The relative lack of bit errors causes the FEC data carried by FX.25 to be redundant. The channel is very noisy at a BER of 10^{-3} and the advantages of the FEC algorithms starts to show, especially with larger frame sizes. AX.25 is barely providing functional throughput at the larger frame sizes, as the frame loss

is substantial. The throughput of FX.25 remains consistent, except with a BER of 10^{-2}, where it struggles to sustain a reasonable throughput with large frame sizes.

The next simulations were done under dynamic channel conditions, with 1 W of transmission power. Four simulations were performed with varying antenna gains to view the effects of reduced signal strength and can be viewed in Figure 4.15.



(a) 0 dB antenna gain.

(b) -3 dB antenna gain.

(c) -4 dB antenna gain.

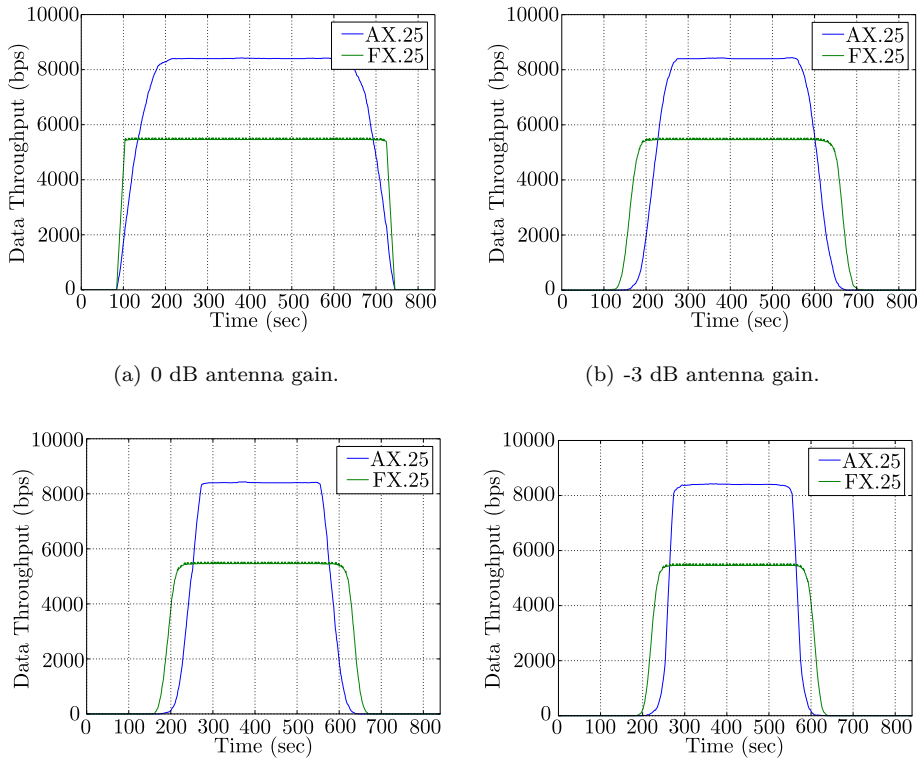
(d) -5 dB antenna gain.

Figure 4.15 – Throughput for noisy channels with decreasing antenna gains.

Figure 4.15(a) shows increased throughput for AX.25, but the two protocols displayed similar throughput with a large data field size. AX.25's throughput remains high due to the motion of the satellite. FX.25 will most likely have a higher throughput than AX.25 when the satellite is far away from the ground station, but AX.25 gets more data through when the satellite is closer to the ground station.

The data throughput for the entire satellite pass is a good metric to analyse the overall protocol performance. It should however be kept in mind that by measuring the data throughput for the entire satellite pass detailed observations are lost, especially considering the dynamics involved with satellite movement. Two aspects which will influence signal strength considerably, is the propagation distance and the antenna gain. Both these parameters change

due to the relative position between the sender and receiver. An antenna will also rarely have a constant gain pattern. It is expected that the throughput should increase as the satellite approaches the ground stations and decrease as it moves further away again. These dynamics need to be studied before an optimal strategy can be developed. The following graphs in Figure 4.16 illustrates the dynamic payload throughput which is calculated over a rolling window of 20 seconds for the entire satellite pass.



(a) 0 dB antenna gain.

(b) -3 dB antenna gain.

(c) -4 dB antenna gain.

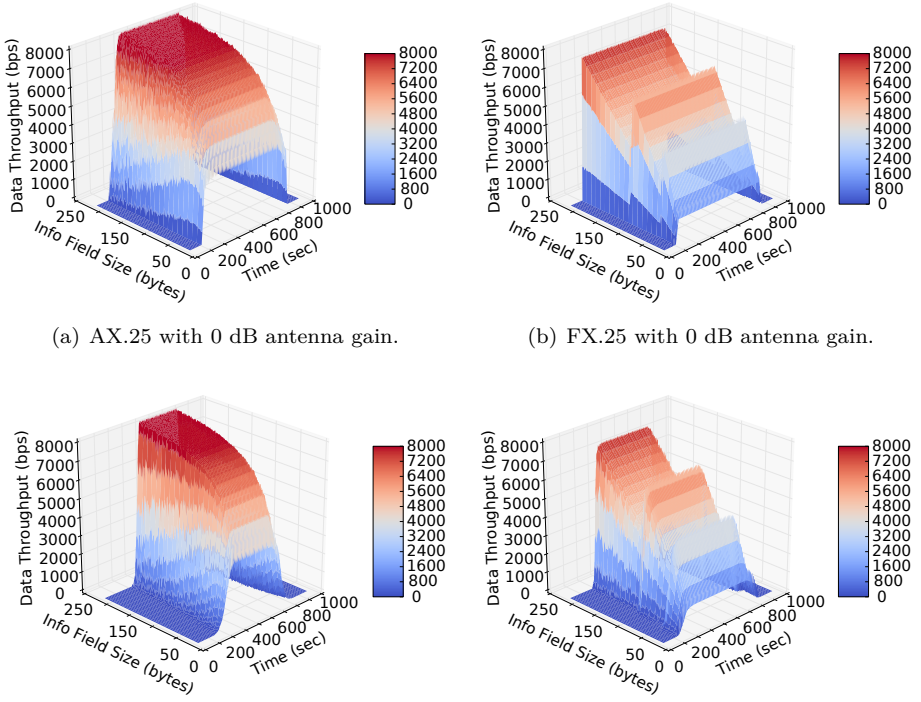
(d) -5 dB antenna gain.

Figure 4.16 – Payload throughput for noisy channels with decreasing antenna gains and a 150 byte info field size (averaged over 10 iterations).

A brief sanity check can be done on the results, to determine if the peak data throughput is realistic. Calculations indicate that the peak throughput should be 8462.85 payload bits per second without any bit stuffing applied. The simulation results have a peak throughput of 8400 payload bits per second. The small difference can be attributed to the effects of bit stuffing.

AX.25 performs very well under 0 dB conditions as can be seen in Figure 4.16(a). The throughput changes as soon as the signal gain reduces. FX.25 is able to transfer data in the early stages of the satellite pass with reduced signal gain, where AX.25 is unable to. AX.25 has a much higher throughput when it is closer to the ground station, as it carries less

overhead data compared to FX.25. Figure 4.17 shows the payload throughput for various info field sizes over the satellite passes.



(a) AX.25 with 0 dB antenna gain.

(b) FX.25 with 0 dB antenna gain.

(c) AX.25 with -4 dB antenna gain.

(d) FX.25 with -4 dB antenna gain.

Figure 4.17 – Payload throughput and info field size for noisy channels with decreasing antenna gains.

Using these existing protocols, it is possible to optimise the throughput by following the line of highest throughput. This requires a protocol that can switch between AX.25 and FX.25. The initial connection can be established with FX.25, which makes it possible for the satellite to communicate earlier than it is able to, using only AX.25. When the amount of bit errors reduce, the communications can switch over to AX.25, thus reducing unnecessary overhead and increasing data throughput.

Another aspect to consider is the choice of RS algorithm to use in FX.25. All the simulations were conducted with worst case results in mind, by using a 16-byte (correct 8 symbols) check value. The 32-byte (correct 16 symbols) and 64-byte (correct 32 symbols) check values provide increased correction ability, which should further increase performance.

The following chapter will investigate a switchable AX/FX.25 protocol stack as an optimised protocol and the potential improvements it can yield.

Chapter 5

Optimal Communication Strategy

5.1 Overview

AX.25 and FX.25 have been assessed extensively, with both protocols showing good performance under specific conditions. To optimise the communication link, it is required to use each protocol under the conditions where it performs best. This involves using FX.25 during the early stages of the satellite pass where the signal is weak and switching over to AX.25 closer to the ground station.

AX.25 can decode FX.25 frames, since the FX.25 protocol is wrapped around a standard AX.25 frame. However, an implementation of FX.25 cannot decode AX.25 frames, as a FX.25 frame is identified by the 64-bit correlation tag and not the AX.25 frame delimiting sequence of 0x7E. This leaves multiple solutions to implementing a modular protocol.

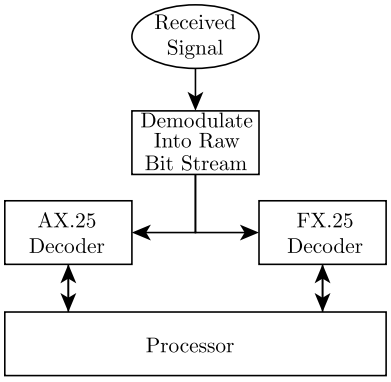


Figure 5.1 – Dual decoding of AX.25 and FX.25.

One approach is illustrated in Figure 5.1. This implementation relies on sharing the demodulated bit stream between an AX.25 and a FX.25 decoder, while feeding both outputs to the processor. The processor can then choose which stream to use. This kind of implementation can be done as a single Field Programmable Gate Array (FPGA) application. This

is however an expensive solution, considering that two decoding processes will be running concurrently and that multiple I/O ports are required on the processor.

A more optimised solution would be a combined AX/FX.25 decoding algorithm, as shown in Figure 5.2, as this can be implemented in hardware as a FPGA based solution, or a pure software implementation. This implementation is more efficient, as it will first attempt to decode the AX.25 part of a FX.25 frame. If this fails, it will decode the full FX.25 frame. The decoding of the FX.25 FEC can be computationally expensive and it may not be required for every frame.

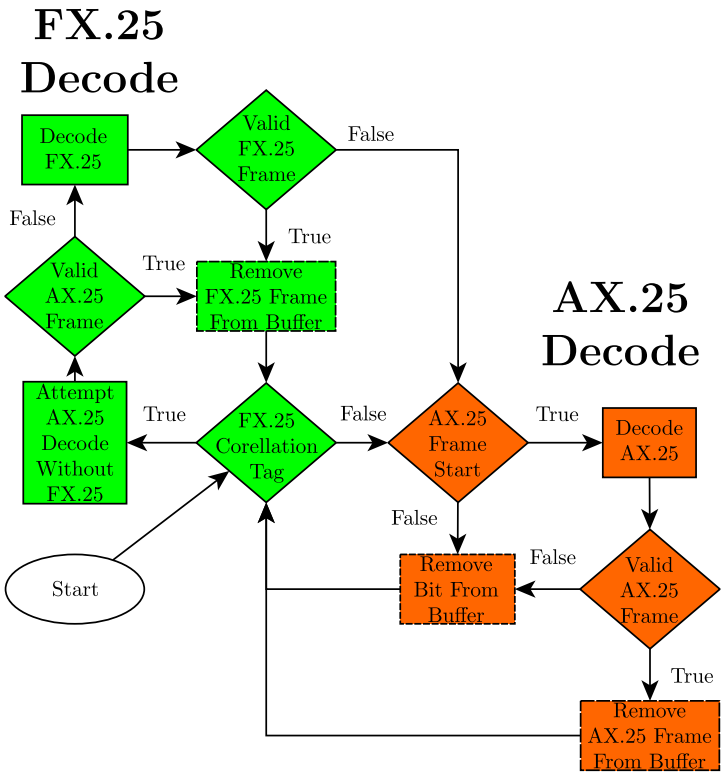


Figure 5.2 – Decoding bit stream flow diagram for modular AX/FX.25.

Another important aspect to consider, is the precise moment when a transmitter should switch between AX.25 and FX.25, since the transmitter will not be aware of any frame loss on the receiver side. A robust solution to this problem is for the receiver to indicate to the transmitter when to switch over from FX.25 to AX.25. With the proposed decoding, shown in Figure 5.2, it is possible to count the number of FX.25 frames received and decoded, without the use of FEC (directly decode the embedded AX.25 frame). By counting these frames and calculating a rolling average, it will be possible to define a threshold where it would be safe to assume that the throughput of AX.25 will be higher. At this point, the receiver will send a frame to the transmitter indicating that it can switch from FX.25 to AX.25.

A similar approach can be used to switch back to FX.25 when the satellite reaches the end of its pass. In this instance, a rolling average of the total number of failed AX.25 decodes without FEC, will be calculated. If this average rises above a threshold, the receiver will indicate to the transmitter that it should switch back to FX.25.

The dynamic protocol switching can be tricky to implement on a half-duplex link, as it requires some form of periodic reporting back to the transmitter. A full duplex-link will allow for a simpler implementation. This type of control can be implemented as an additional layer on top of the hybrid AX/FX.25 protocol.

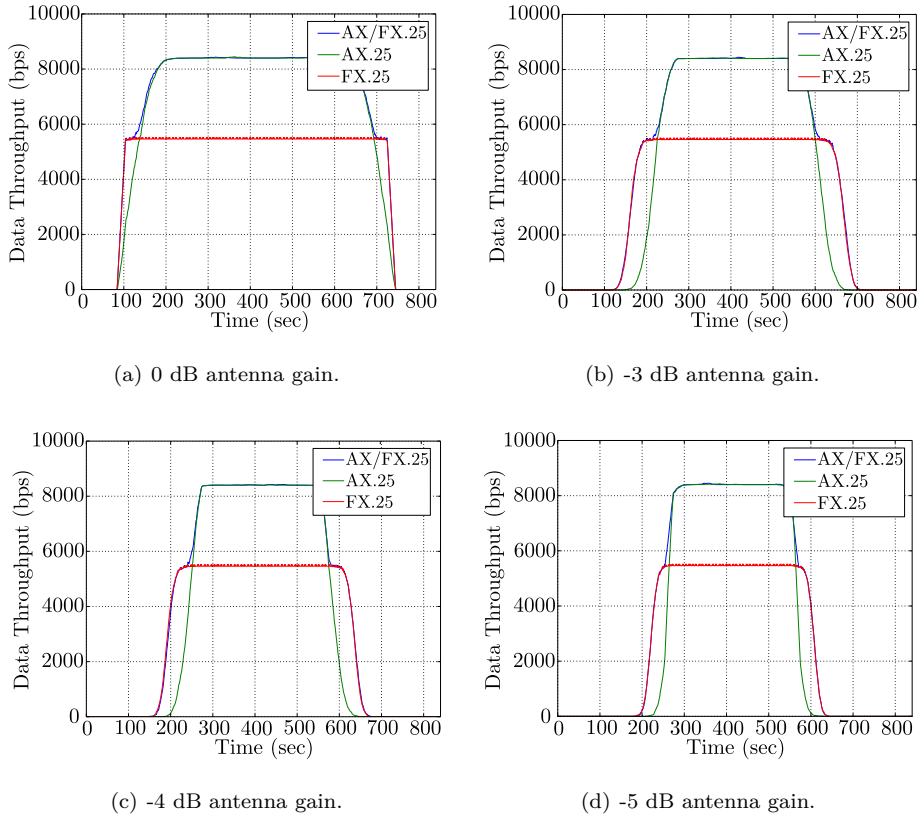
5.2 Implementation

The proposed hybrid decoding (as in Figure 5.2) was implemented in SatSim to test the logic and measure the potential throughput performance. This implementation is only for simultaneous decoding of AX.25 and FX.25 frames as in Figure 5.2. A transmitter was not developed, as only the decoder is required to simulate the performance of the hybrid strategy.

5.3 Results

Simulation configurations similar to those conducted in Section 4.4 were executed, to test and analyse the hybrid decoding implementation. The expected throughput behaviour should be similar to the results achieved in Figure 4.16, but with the hybrid decoder following the envelope of the highest throughput. Two separate transmitters were used in the simulation, one transmitting AX.25 frames and the other FX.25, to emulate an AX/FX.25 transmitter. The simulation was configured to switch between these two transmitters in a time scheduled routine during the satellites pass. The switching times were determined from the results in Figure 4.16.

The results for the hybrid switching protocol can be viewed in Figure 5.3. A first observation would be that the hybrid protocol does not exactly follow the envelope of highest throughput, but this is due to the effect of the rolling window calculation of the throughput. The implementation of the decoder functions as expected, as it dynamically decoded any incoming AX.25 and FX.25 frames.



(a) 0 dB antenna gain.

(b) -3 dB antenna gain.

(c) -4 dB antenna gain.

(d) -5 dB antenna gain.

Figure 5.3 – Payload throughput for noisy channels with decreasing antenna gains, a 150 byte info field size and 16-byte FEC check values (averaged over 10 iterations).

Table 5.1 summarises the number of payload data bits received by each receiver and Table 5.2 shows the improvement of the hybrid decoder over standard AX.25 and FX.25. The improvement over FX.25 is substantial, as the ability to switch to AX.25 allows the exploitation of the increased throughput of AX.25. The improvement on AX.25 is also significant as soon as the signal gain is reduced. A throughput increase of at least 14% is expected when the hybrid AX/FX.25 is applied.

Table 5.1 – Protocol throughput for various antenna gains with a 150 byte info field size and a 16-byte FEC check value (averaged over 10 iterations).

	Protocol Payload Throughput (bits)		
Antenna Gain	AX.25	FX.25	AX/FX.25
0 dB	4 835 160	3 506 760	5 017 920
-3 dB	3 294 240	2 769 960	3 779 640
-4 dB	2 884 560	2 431 560	3 326 520
-5 dB	2 607 960	2 118 960	2 973 240

Table 5.2 – Protocol throughput improvement for various antenna gains with a 150 byte info field size and a 16-byte FEC check value (averaged over 10 iterations).

	AX/FX.25 Payload Throughput Improvement	
Antenna Gain	AX.25	FX.25
0 dB	3.78 %	43.09 %
-3 dB	14.73 %	36.45 %
-4 dB	15.32 %	36.81 %
-5 dB	14.01 %	40.32 %

The increase in payload throughput is a key advantage of the hybrid protocol, but it is also worth noting the improved reliability of the communication link. The FX.25 segment also increases access times, as communication can be established at lower elevations. These simulations were conducted for worst case improvement in throughput performance, as the RS FEC was selected with a 16-byte check value, which enables the correction of only 8 symbols.

As mentioned in Section 4.4, the 32-byte and 64-byte check values should provide a further performance increase. The stronger check values were investigated under the same test conditions as in this chapter, but with the pure FX.25 implementation also making use of the stronger check value. The detailed results can be viewed in Appendix D. The hybrid protocol with a 32-byte check value showed a throughput increase of at least 21% over AX.25 and 33% over FX.25. The hybrid protocol with a 64-byte check value provided a throughput increase of at least 25% over AX.25 and 31% over FX.25.

Chapter 6

Conclusion, Recommendation and Future Work

6.1 Conclusion

The two objectives of this thesis were the development of an innovative network simulation tool and an investigation into an optimal network protocol strategy. SatSim, the developed simulation tool in Python, proved capable of simulating and analysing network protocols. The design drivers for SatSim were user-friendliness, modularity and re-usability. These aspects were identified after a brief investigation into the current state of network simulation tools in the industry. SatSim was verified by comparing simulation results to previous work and theoretical calculations.

The second objective, of investigating an optimal network protocol, was also achieved. This investigation started with an analysis of the current trends in CubeSat development procedures regarding radio communications. It was concluded that the restrictions of the CubeSat specification limit the capabilities of the radio communication hardware deployed on missions. Thus, it was decided to investigate improvements on a network protocol level. AX.25 is one of the most common protocols implemented with CubeSat missions and was a logical initial choice for implementation in SatSim. Simulation results of AX.25 were used as a baseline to improve upon.

AX.25 has some limitations when applied to space based applications, due to its origins in the amateur radio community. The most significant limitation being the lack of any form of FEC. FX.25, developed by the Stensat group in 2005, addresses this shortcoming, by adding a FEC wrapper around the AX.25 frame, while maintaining backward compatibility with an AX.25 decoder. FX.25 was also implemented in SatSim. The simulation results showed that FX.25 can communicate more reliably and at lower elevations than AX.25 under the same channel conditions, but at a reduced payload throughput. At this point, it became clear that a hybrid protocol solution could provide the optimal solution if it could exploit the unique characteristics of both protocols. FX.25 can be used at the start of a satellite pass, as it can operate at lower elevations than AX.25. A switch over to AX.25 can then take

place to increase throughput, when the satellite is closer to the ground station (increased signal gain).

A hybrid protocol was implemented and simulated in SatSim, to prove the feasibility and test the performance increase of such a strategy over regular AX.25 and FX.25. Simulation results confirmed an increased throughput of at least 14% over AX.25 and 36% over FX.25, under the defined test configuration. These are worst case results, as the weakest FEC was chosen for the simulations. Further improvements can be achieved, by increasing the strength of the FEC algorithm. Some practical aspects concerning the implementation of such a hybrid protocol was also discussed, which included how to intelligently approach the protocol switching during communications.

6.2 Future Work and Recommendation

SatSim version 1.0 includes all the functionality required to complete this thesis. As with any software development project, there is always room for improvement and additions, especially with a scientific tool, such as SatSim. The limitations and potential future work regarding SatSim are discussed in Chapter 3.

The hybrid AX/FX.25 protocol implemented in SatSim is currently only a decoder. It would be of interest to simulate a full hybrid protocol, including a transmitter. This would provide an opportunity to test and assess the suggested full AX/FX.25 implementation with the intelligent switching capability, as discussed in Section 5.1.

Another area of future work that reaches beyond the simulation environment, is the implementation of the hybrid AX/FX.25 protocol in a software defined radio environment or FPGA (or both). This could stimulate the development of a new hardware unit for the CubeSat community.

Appendix A

CubeSat Missions Summary

Table A.1 – A summary of CubeSat communication hardware as adapted from Ref. [12] (part 1 of 5).

Launch	Satellite	Size	Frequency	Power	Protocol	Data Rate/Modulation	Downloaded	Life Time	Antenna
30 June 2003	AAU1	1U	437.475 MHz	500 mW	AX.25, Mobitex	9600 Baud GMSK	1kB	3 Months	Dipole
	DTUsat-1	1U	437.475 MHz	400 mW	AX.25	2400 Baud FSK	0	0 days	Canted Turnstile
	CanX-1	1U	437.880 MHz	500 mW	Custom	1200 Baud MSK	0	0 days	Crossed Dipoles
	Cute-1	1U	437.470 MHz	350 mW	AX.25	1200 Baud AFSK	>10 MB	118+ Months	Monopole
	QuakeSat-1	3U	436.675 MHz	2 W	AX.25	9600 Baud FSK	423 MB	7 Months	Turnstile
	XI-IV	1U	437.490 MHz	1 W	AX.25	1200 Baud AFSK	>11 MB	118+ Months	Dipole
27 Oct 2005	XI-V	1U	437.345 MHz	1 W	AX.25	1200 Baud AFSK	N/A	90+ Months	Dipole
	NCube-2	1U	437.505 MHz	N/A	AX.25	1200 Baud AFSK	0	0 days	Monopole
	UWE-1	1U	437.505 MHz	1 W	AX.25	1200/9600 Baud AFSK	N/A	3 Weeks	Dipole
22 Feb 2006	Cute-1.7	2U	437.505 MHz	300 mW	AX.25, SRSLL	1200 AFSK / 9600 GMSK	<1MB	2.5 Months	Dipole
11 Dec 2006	GeneSat-1	3U+	2.4 GHz	1 W	Proprietary	N/A	500kB	3 Months	Patch
17 Apr 2007	CSTB1	1U	400.0375 MHz	300 mW	Proprietary	1200 Baud AFSK	6.77MB	18 Months	Dipole
	AeroCube-2	1U	915 MHz	2 W	Proprietary	38.4 kBaud	500kB	1 Week	Patch
	CP4	1U	437.325 MHz	1 W	AX.25	1200 Baud FSK	487kB	2 Months	Dipole
	Libertad-1	1U	437.405 MHz	400 mW	AX.25	1200 Baud AFSK	0	1 Month	Monopole
	CAPE1	1U	435.245 MHz	1 W	AX.25	9600 Baud FSK	0	4 Months	Dipole
	CP3	1U	436.845 MHz	1 W	AX.25	1200 Baud FSK	2MB	19+ Months	Dipole
	MAST	3U	2.4 GHz	1 W	Proprietary	15 kbps	>2MB	0.75 Months	Monopole
28 Apr 2008	Delfi-C3	3U	435.55 MHz	400 mW	AX.25	1200 Baud BPSK	60MB	60+ Months	Turnstile
	Seeds-2	1U	437.485 MHz	450 mW	AX.25	1200 Baud AFSK	500kB	60+ Months	Monopole
	CanX-2	3U	2.2 GHz	500 mW	NSP	16-256kbps BPSK	250MB	60+ Months	Patch
	AAUSAT-II	1U	437.425 MHz	610 mW	AX.25	1200 Baud MSK	8MB	60+ Months	Dipole
	Compass-1	1U	437.405 MHz	300 mW	AX.25	1200 Baud AFSK/MSK	<1MB	60+ Months	Dipole
19 May 2009	AeroCube-3	1U	915 MHz	2 W	Proprietary	77 kBaud GFSK	52MB	7 Months	Patch
	CP6	1U	437.365 MHz	1 W	AX.25	1200 Baud FSK	N/A	4 Months	Dipole
	HawkSat-1	1U	437.345 MHz	1 W	Proprietary	N/A	0	0 Days	Monopole
	PharmaSat	3U+	2.4 GHz	1 W	Proprietary	10kbps	650 kB	10 Days	Patch
23 Sept 2009	BEESAT-1	1U	436.000 MHz	500 mW	Mobitex	4800/9600 Baud GMSK	N/A	43+ Months	Dipole
	UWE-2	1U	437.385 MHz	N/A	AX.25	1200 Baud AFSK	N/A	1 Week	Dipole
	ITUpSAT-1	1U	437.325 MHz	1W	Proprietary	19200 Baud	0	43+ Months	Dipole
	SwissCube	1U	437.505 MHz	1W	AX.25	1200 Baud FSK	6MB	43+ Months	Monopole
20 May 2010	Hayato	1U	13.275 GHz	100 mW	N/A	10kbps/1Mbps BPSK	0	18 Days	Patch
	Waseda-SAT2	1U	437.485 MHz	150 mW	AX.25	9600 Baud FSK	0	0 Days	Monopole
	Negai-Star	1U	437.305 MHz	150 mW	AX.25	1200 Baud FSK	N/A	1 Months	Dipole
12 July 2010	Tisat-1	1U	437.305 MHz	500 mW	AX.25	1200 Baud AFSK	N/A	33+ Months	Monopole
	StudSat	1U	437.505 MHz	500 mW	Custom AX.25	4800 Baud FSK	0	5 days	Monopole
19 Nov 2010	RAX-1	3U	437.505 MHz	750 mW	AX.25	9600 Baud GMSK	4.8MB	2 Months	Turnstile
	O/OREOS	3U+	2.4 GHz	1 W	Proprietary	Variable	8MB	29+ Months	Patch
	NanoSail-D2	3U+	2.4 GHz	1 W	Proprietary	Variable	N/A	5 Days	Patch
8 Dec 2010	Perseus	1.5U	N/A	N/A	N/A	N/A	N/A	1 Month	Dipole
	QbX	3U	450 MHz	1 W	N/A	9600 Baud GMSK	N/A	1 Months	Quadrafilar Helix
	SMDC-ONE	3U	UHF	N/A	N/A	N/A	N/A	1 Months	Turnstile
	Mayflower	3U	437.000 MHz	1 W	Proprietary	Variable	0	2 Days	Dipole

Table A.2 – A summary of CubeSat communication hardware as adapted from Ref. [12] (part 2 of 5).

Launch	Satellite	Size	Frequency	Power	Protocol	Data Rate/Modulation	Downloaded	Life Time	Antenna
12 Oct 2011	Jugnu	3U	437.505 MHz	1 W	AX.25	2400 Baud FSK	N/A	18+ Months	Monopole
28 Oct 2011	AubieSat-1	1U	437.475 MHz	800 mW	CW	20 WPM	0	18+ Months	Dipole
	DICE	1.5U	465 MHz	1 W	Proprietary	1.5 Mbps BPSK	N/A	18+ Months	Dipole
	HRBE	1U	437.505 MHz	850 mW	AX.25	1200 Baud FSK	N/A	18+ Months	Monopole
	M-Cubed	1U	437.485 MHz	1 W	AX.25	1200 Baud FSK	0	18+ Months	Monopole
	RAX-2	3U	437.345 MHz	1 W	AX.25	9600 Baud GMSK	242MB	18+ Months	Turnstile
13 Feb 2012	Xatcobeo	1U	437.365 MHz	500 mW	AX.25	1200 Baud MSK	N/A	14+ Months	Turnstile
	ROBUSTA	1U	437.325 MHz	800 mW	AX.25	1200 Baud AFSK	0	2 Days	Dipole
	e-st@r	1U	437.445 MHz	500 mW	AX.25	1200 Baud AFSK	0	3 Days	Dipole
	Goliat	1U	2.4 GHz	1 W	Proprietary	Variable	0	1 Week	Patch
	PW-Sat	1U	145.9 MHz	200 mW	AX.25	1200 Baud BPSK	N/A	10 Months	Dipole
	Masat-1	1U	437.345 MHz	100/400 mW	Custom	GFSK	305MB	14+ Months	Monopole
	UniCubeSat-GG	1U	437.305 MHz	500 mW	AX.25	9600 Baud GFSK	0	2 Days	Dipole
13 Sept 2012	SMDC-ONE	3U	UHF	N/A	N/A	N/A	N/A	N/A	Turnstile
	AeroCube-4	1U	915 MHz	2 W	Proprietary	38.4 kBaud	N/A	8+ Months	Patch
	Aeneas	3U	437.000 MHz	1 W	Proprietary	Variable	N/A	8+ Months	Monopole
	CSSWE	3U	437.345 MHz	1 W	AX.25	9600 Baud GFSK	60MB	8+ Months	Monopole
	CP5	1U	437.405 MHz	500 mW	AX.25	1200 Baud FSK	500kB	4 Months	Dipole
	CXBN	2U	437.525 MHz	1 W	AX.25	9600 Baud GFSK	N/A	8+ Months	Turnstile
	CINEMA	3U	2.2 GHz	1 W	Proprietary	1 Mbps FSK	N/A	8+ Months	Patch
	Re	3U	915 MHz	1 W	AX.25	57.6 kbps FSK	N/A	N/A	Dipole
4 Oct 2012	FITSat-1	1U	5.84 GHz	2 W	N/A	115.2 kbps FSK	N/A	9 Months	Patch
			437.445 MHz	N/A	AX.25	1200 Baud AFSK	N/A	9 Months	Monopole
	TechEdSat	1U	437.465 MHz	1 W	AX.25	1200 Baud AFSK	N/A	7 Months	N/A
	F1	1U	145.980 MHz	1 W	AX.25	1200 Baud AFSK	0	0 Days	Dipole
	WE-WISH	1U	437.505 MHz	10 0mW	SSTV	30kbps SSTV	N/A	5 Months	Monopole
	RAIKO	2U	13 GHz	N/A	N/A	N/A	N/A	10 Months	Patch
25 Feb 2013	STRaND-1	3U	437.575 MHz	1.6 W	AX.25	9600 Baud FSK	N/A	6+ Months	Turnstile
	AAUSAT3	1U	437.425 MHz	900 mW	CCSDS	2400 bps	N/A	6+ Months	Turnstile
19 April 2013	OSSI-1	1U	437.525 MHz	2 W	AX.25	9600 Baud FSK	0	0 Days	Dipole
	SOMP	1U	437.485 MHz	500 mW	Custom	23 kbps BPSK	N/A	2+ Weeks	Turnstile
	BEESAT-2	1U	435.950 MHz	500 mW	Mobitex	4800 bps GMSK	N/A	2+ Weeks	Monopole
	BEESAT-3	1U	435.95 MHz	500 mW	Mobitex	4800 bps GMSK	N/A	2+ Weeks	Monopole
	Dove-2	3U	401.3 MHz	1.6 W	N/A	2.4 kbps FSK	N/A	4+ Months	Monopole
21 April 2013	Phonesat 1 Graham	1U	437.425 MHz	1 W	AX.25	1200 Baud FSK	N/A	1 Week	Monopole
	Phonesat 1 Bell	1U+	437.425 MHz	1 W	AX.25	1200 Baud FSK	N/A	1 Week	Monopole
	Phonesat 2b Alexander	1U	437.425 MHz	1 W	AX.25	1200 Baud FSK	N/A	1 Week	Monopole
	Dove-1	3U	401.3 MHz	1.6 W	N/A	2.4 kbps FSK	N/A	1 Week	Monopole
		3U	8.225 GHz	3 W	IP over DVB-S2	4Mbps QPSK	N/A	1 Week	Patch
19 April 2013	TurkSat-3USat	3U	435.225 MHz	N/A	N/A	50kHz	N/A	1 Week	Monopole
			437.225 MHz	1 W	AX.25	9600 Baud FSK	N/A	1 Week	Monopole
	CubeBug-1	2U	437.445 MHz	1 W	AX.25	1200 Baud AFSK	N/A	4+ Months	Turnstile
	NEE-01 Pegasus	1U	910 MHz	1.9 W	SSTV/Audio	N/A	N/A	1 Month	N/A
7 May 2013	ESTCube-1	1U	437.505 MHz	500 mW	AX.25	9600 Baud FSK	N/A	3+ Months	Monopole

Table A.3 – A summary of CubeSat communication hardware as adapted from Ref. [12] (part 3 of 5).

Launch	Satellite	Size	Frequency	Power	Protocol	Data Rate/Modulation	Downloaded	Life Time	Antenna
19 Nov 2013	ArduSat-1	1U	437 MHz	1 W	CSP	9600 Baud FM-MSK		2+ Months	Turnstile
	ArduSat-X	1U	437 MHz	1 W	CSP	9600 Baud FM-MSK		2+ Months	Turnstile
	Pico Dragon	1U	437.365 MHz 437.250 MHz	800 mW 100 mW	AX.25 CW	1200 Baud AFSK		1 Month	Monopole Monopole
	TechEdSat-3P	3U	437.465 MHz 1616 MHz	1 W 1 W	AX.25 Proprietary	1200 Baud AFSK	N/A	1 Month	Monopole Patch
20 Nov 2013	COPPER	1U	437.29 MHz	1 W	AX.25	9600 Baud FSK			Dipole
	TJ3Sat	1U	437.32 MHz	1 W	AX.25	1200 Baud AFSK			
	Vermont Lunar Cube	1U	437.305 MHz	1.5 W	AX.25	9600 Baud FSK		1+ Months	Crossed Dipole
	SwampSat	1U	437.385 MHz	1 W	AX.25	9600 Baud FSK			
	CAPE-2	1U	437.325 MHz 145.825 MHz		Parrot Repeater	FM	N/A	1+ Months	
	Ho'oponopono-2	3U	2.4 GHz 437.22 MHz	1 W	Proprietary	57.6 kbps			Patch Monopole
	PhoneSat-v2.4	1U	437.425 MHz	1 W	AX.25	1200 Baud AFSK			Monopole
	Trailblazer	1U	437.425 MHz	1 W	AX.25	9600 Baud FSK			Turnstile
	DragonSat-1	1U	145.87 MHz	500 mW	AX.25	9600 Baud FSK			Monopole
	KySat-2	1U	437.405 MHz	1.5 W	AX.25	9600 Baud FSK		1+ Months	Turnstile
	ChargerSat-1	1U	437.405 MHz	1 W		9600 Baud FSK			Dipole
	NPS-SCAT	1U	2.4 GHz 437.525 MHz	1 W 1 W	Proprietary AX.25	1200 Baud FSK	N/A		Patch Dipole
	Black Knight 1	1U	437.345 MHz	1 W	Proprietary				
	ORS A	3U							
	ORS B	3U							
	Prometheus	1.5U							
	SENSE	3U	2.2 GHz						Patch
	FireFly	3U	425 MHz	1 W	Proprietary	QPSK			Dipole
	Horus	3U	915 MHz						Patch
21 Nov 2013	FUNcube-1	1U	145.936 MHz	300 mW		1200 Baud BPSK	70MB+	2+ Months	Dipole
	ZACube-1	1U	437.345 MHz		AX.25	1200 Baud AFSK/9600 Baud FSK		2+ Months	Dipole
	HiNcube	1U	437.305 MHz		AX.25	9600 Baud GMSK			
	First-MOVE	1U	145.970 MHz	230 mW	AX.25	1200 Baud BPSK		2+ Months	Dipole
	UWE-3	1U	437.385 MHz		AX.25	9600 Baud FSK		2+ Months	
	Velox-PII	1U	145.980 MHz	230 mW	AX.25	1200 Baud BPSK		2+ Months	
	NEE-02 KRYSAOR	1U	980 MHz						
	CubeBug-2	2U	437.445 MHz	1 W	AX.25	1200/9600 Baud AFSK/GFSK		2+ Months	Turnstile
	KHUSAT	3U	2.2 GHz	1 W	Proprietary	1 Mbps FSK		2+ Months	Patch
	TRITON-1	3U	145.815 MHz	230 mW	AX.25	9600 Baud BPSK		2+ Months	Dipole
	Delfi-n3xt	3U	145. 9MHz 2.405 GHz	230 mW 125 mW	AX.25	9600 Baud BPSK 20-500 kbps MSK		2+ Months	Dipole Patch
	OPTOS	3U+	402 MHz						
	Dove-3	1U	401.3 MHz 8.1 GHz	1.6 W 3 W	IP over DVB-S2				Monopole Patch
	PUCP-SAT-1	1U	145.84 MHz 437.2 MHz	1.5 W 10 mW	AX.25 CW	1200 Baud AFSK 12 WPM	N/A		Dipole Dipole
	ICUBE-1	1U	145.947 MHz	23 mW	AX.25	1200 Baud BPSK			
	HumSAT-D	1U	437.325 MHz	500 mW	CSP/CW	1200 Baud MSK		2+ Months	Turnstile
	Dove-4	3U+	401.3 MHz 8.1 GHz	1.6 W 3 W	IP over DVB-S2				Monopole Patch

Table A.4 – A summary of CubeSat communication hardware as adapted from Ref. [12] (part 4 of 5).

Launch	Satellite	Size	Frequency	Power	Protocol	Data Rate/Modulation	Downloaded	Life Time	Antenna
5 Dec 2013	IPEX	1U	437.27 MHz	1 W	AX.25	9600 Baud FSK		2+ Months	Dipole
	MCubed-2	1U	437.485 MHz	1 W	AX.25	9600 Baud GMSK		2+ Months	Monopole
	CUNYSAt-1	1U	437.505 MHz	1.2 W					
	FIREBIRD	1.5U	437.405 MHz	1 W			27MB+	2+ Months	Dipole
	Alice	3U							
	AeroCube-5	1.5U	915 MHz						
	SMDC-ONE	3U							
	TacSat-6	3U							
	SNAP	3U							
11-28 Feb 2014	Flock-1	3U+	401.3 MHz 8.100 GHz	1.6 W 3 W	IP over DVB-S2				Monopole Patch
	UAPSAT	1U	437.385 MHz		AX.25				Turnstile
	ArduSat-2	2U	400 MHz 2.4 GHz	2 W 2.3 W					Turnstile Patch
	SkyCube	1U	915 MHz		AX.25	57.6 kbps			Dipole
	LitSat-1	1U	145.85 MHz		AX.25				
	LituanicaSAT-1	1U							
27 Feb 2014	INVADER	1U	437.2 MHz 437.325 MHZ	800 mW 100 mW	AX.25 CW	1200 Baud AFSK		2+ Months	Dipole Dipole
	KSAT-2	1U	S-band Ku-band						
	OPUSAT	1U	437.15 MHz		CW/AX.25	1200 Baud AFSK/9600 GMSK		2+ Months	
18 Apr 2014	SporeSat	3U	437.1 MHz	1 W	AX.25	1200 Baud AFSK		Deorbited	Monopole
	TSAT	2U	1.6 GHz	200 mW	Proprietary	36 bytes/sec		Deorbited	Patch
	PhoneSat-v2.5	1U	437.425 MHz		AX.25	1200 Baud AFSK		Deorbited	Monopole
	ALL-STAR	3U	2401.7 MHz					Deorbited	
	KickSat	3U	437.505 MHz	1 W	AX.25	1200 Baud AFSK		Deorbited	
19 June 2014	AntelSat	2U	437.280 MHz 437.575 MHz 2403.00 MHz	200 mW 1 W 1 W	CW AX.25 AX.25	1200 Baud FSK/AFSK 500 kbps GFSK		1.5 Months	Monopole Monopole Patch
	AeroCube-6	0.5U	915 MHz	1.3 W	Proprietary	500 kbps FSK		1.5 Months	Patch
	LEMUR-1	3U	402 MHz	2 W					
	TigriSat	3U							
	Flock-1	3U+	401.3 MHz 8.1 GHz	1.6 W 3 W	IP over DVB-S2	40 Mbps		1.5 Months	Monopole Patch/Helix
	Perseus	6U	401 MHz 26.8 GHz	1 W 500 mW	AX.25 DVB-S2	40 Mbps		1.5 Months	Monopole Horn
	QB50P1	2U	145.815 MHz 145.950 MHz	200 mW 400 mW	CW/AX.25	15 WPM/ 1200 Baud BPSK Linear transponder		1.5 Months	Dipole Dipole
	QB50P2	2U	145.880 MHz 145.940 MHz	200 mW	CW/AX.25 FX.25	15 WPM/1200 Baud BPSK 9600 bps FSK		1.5 Months	Dipole Dipole
	DTUSat-2	1U	2.401 GHz	220 mW	CW/FSK				
	PolyITAN-1	1U	437.675 MHz	600 mW	CW/AX.25	9600 Baud FSK			Dipole
	Duchifat-1	1U	145.980 MHz 145.825 MHz	200 mW	CW/AX.25 AX.25	15 WPM/1200 Baud BPSK 1200 Baud AFSK		1.5 Months 1.5 Months	Dipole Dipole
	NanoSatC-Br 1	1U	145.835 MHz	200 mW	CW/AX.25	15 WPM/1200 Baud BPSK		1.5 Months	Dipole
	PACE	2U	437.485 MHz	1 W	CW/AX.25	15 WPM/1200 Baud FSK			Dipole
	POPSAT-HIP-1	3U	437.405 MHz		CCSDS				

Table A.5 – A summary of CubeSat communication hardware as adapted from Ref. [12] (part 5 of 5).

Launch	Satellite	Size	Frequency	Power	Protocol	Data Rate/Modulation	Downloaded	Life Time	Antenna
30 June 2014	VELOX-1	3U	145.980 MHz	200 mW	AX.25	1200 Baud BPSK			
8 July 2014	UKube-1	3U	145.840 MHz 145.915 MHz 2.401 GHz	200 mW 300 mW 1 W	AX.25	1200 Baud BPSK 1200 Baud BPSK 2 Mbps QPSK			Dipole Dipole Patch

Appendix B

BCH Coding Table

Table B.1 – BCH modulation table. n = block length, k = data bits, t = correctable bits. Adapted from Ref. [13].

n	**k**	**t**	**n**	**k**	**t**	**n**	**k**	**t**
255	187	9	511	322	22	1023	933	9
	179	10		313	23		923	10
	171	11		304	25		913	11
	163	12		295	26		903	12
	155	13		286	27		893	13
	147	14		277	28		883	14
	139	15		268	29		873	15
	131	18		259	30		863	16
	123	19		250	31		858	17
	115	21		241	36		848	18
	107	22		238	37		838	19
	99	23		229	38		828	20
	91	25		220	39		818	21
	87	26		211	41		808	22
	79	27		202	42		798	23
	71	29		193	43		788	24
	63	30		184	45		778	25
	55	31		175	46		768	26
	47	42		166	47		758	27
	45	43		157	51		748	28
	37	45		148	53		738	29
	29	47		139	54		553	52
	21	55		130	55		543	53
	13	59		121	58		533	54
	9	63		112	59		523	55
511	502	1		103	61		513	57
	493	2		94	62		503	58
	484	3		85	63		493	59
	475	4		76	85		483	60
	466	5		67	87		473	61
	457	6		58	91		463	62
	448	7		49	93		453	63
	439	8		40	95		443	73
	430	9		31	109		268	103
	421	10		28	111		258	106
	412	11		19	119		249	107
	403	12		10	121		238	109
	394	13	1023	1013	1		228	110
	385	14		1003	2		218	111
	376	15		993	3		208	115
	367	16		983	4		203	117
	358	18		973	5		193	118
	349	19		963	6		183	119
	340	20		953	7		173	122
	331	21		943	8		163	123

Appendix C

Additional AX.25 Information

Table C.1 – Example non-repeater AX.25 I-frame. From N7LEM (SSID=0) to NJ7P (SSID=0) without a Layer 3 protocol. The P/F bit is set. The receive sequence number, N(R), is 1 and the send sequence number, N(S), is 7. Adapted from Ref. [10].

Byte	ASCII	Bin Data	HEX Data
Flag		01111110	7E
A1	N	10011100	98
A2	J	10010100	94
A3	7	01101110	6E
A4	P	10100000	A0
A5	space	01000000	40
A6	space	01000000	40
A7	SSID	11100000	E0
A8	N	10011100	98
A9	7	01101110	6E
A10	L	10011000	98
A11	E	10001010	8A
A12	M	10011010	9A
A13	space	01000000	40
A14	SSID	01100001	61
Control	I	001111110	3E
PID	none	11110000	F0
FCS	part1	XXXXXXXX	HH
FCS	part2	XXXXXXXX	HH
Flag		01111110	7E

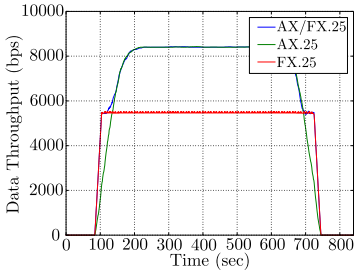
Table C.2 – PID field definitions. "y" Indicates all combinations. Adapted from Ref. [10].

HEX	Bin Data	Translation
**	yy01yyyy	AX.25 layer 3 implemented.
**	yy10yyyy	AX.25 layer 3 implemented.
0x01	00000001	ISO 8208/CCITT X.25 PLP.
0x06	00000110	Compressed TCP/IP packet. Van Jacobson (RFC 1144)
0x07	00000111	Uncompressed TCP/IP packet. Van Jacobson (RFC 1144)
0x08	00001000	Segmentation fragment
0xC3	11000011	TEXNET datagram protocol
0xC4	11001011	Link Quality Protocol.
0xCA	11001010	Appletalk
0xCB	11001011	Appletalk ARP
0xCC	11001100	ARPA Internet Protocol
0xCD	11001101	ARPA Address resolution
0xCE	11001110	FlexNet
0xCF	11001111	Net/ROM
0xF0	11110000	No layer 3 protocol implemented.
0xFF	11111111	Escape character. Next byte contains more Level 3 protocol information.
Escape character. Next byte contains more Level 3 protocol information.	00001000	

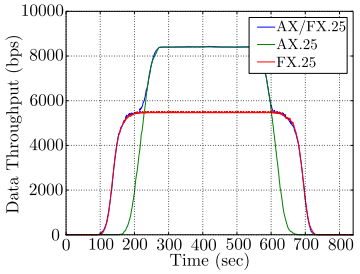
Appendix D

Additional Hybrid AX/FX.25 Results

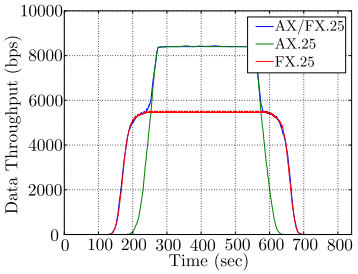
D.1 32-Byte Check Value



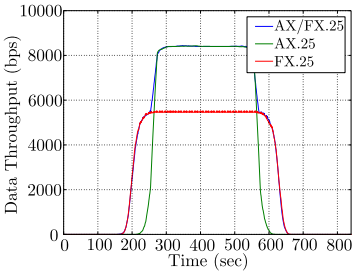
(a) Payload throughput for 0 dB antenna gain.



(b) Payload throughput for -3 dB antenna gain.



(c) Payload throughput for -4 dB antenna gain.



(d) Payload throughput for -5 dB antenna gain.

Figure D.1 – Payload throughput for noisy channels with decreasing antenna gains, a 150 byte info field size and 32-byte FEC check values (averaged over 10 iterations).

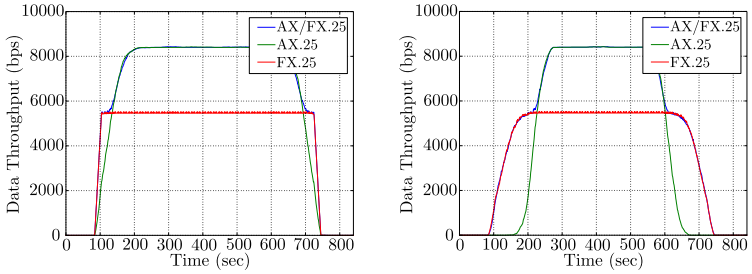
Table D.1 – Protocol throughput for various antenna gains with a 150 byte info field size and a 32-byte FEC check value (averaged over 10 iterations).

	Protocol Payload Throughput (bits)		
Antenna Gain	AX.25	FX.25	AX/FX.25
0 dB	4 845 840	3 506 760	5 011 800
-3 dB	3 289 200	3 004 560	4 009 920
-4 dB	2 878 080	2 637 360	3 543 720
-5 dB	2 608 920	2 313 000	3 167 640

Table D.2 – Protocol throughput improvement for various antenna gains with a 150 byte info field size and a 32-byte FEC check value (averaged over 10 iterations).

	AX/FX.25 Payload Throughput Improvement	
Antenna Gain	AX.25	FX.25
0 dB	3.42 %	42.92 %
-3 dB	21.91 %	33.46 %
-4 dB	23.13 %	34.37 %
-5 dB	21.42 %	36.95 %

D.2 64-Byte Check Value



(a) Payload throughput for 0 dB antenna gain.

(b) Payload throughput for -3 dB antenna gain.

(c) Payload throughput for -4 dB antenna gain.

(d) Payload throughput for -5 dB antenna gain.

Figure D.2 – Payload throughput for noisy channels with decreasing antenna gains, a 150 byte info field size and 64-byte FEC check values (averaged over 10 iterations).

Table D.3 – Protocol throughput for various antenna gains with a 150 byte info field size and a 64-byte FEC check value (averaged over 10 iterations).

	Protocol Payload Throughput (bits)		
Antenna Gain	AX.25	FX.25	AX/FX.25
0 dB	4 845 840	3 507 120	5 018 760
-3 dB	3 298 680	3 159 000	4 161 840
-4 dB	2 890 320	2 778 840	3 674 400
-5 dB	2 613 840	2 437 800	3 288 600

Table D.4 – Protocol throughput improvement for various antenna gains with a 150 byte info field size and a 64-byte FEC check value (averaged over 10 iterations)..

	AX/FX.25 Payload Throughput Improvement	
Antenna Gain	AX.25	FX.25
0 dB	3.50 %	43.10 %
-3 dB	26.17 %	31.75 %
-4 dB	27.13 %	32.23 %
-5 dB	25.81 %	34.90 %

Bibliography

[1] Kearney, M.A., Botma, P., Jordaan, W., Gerber, J., Thesnaar, E., Nolte, F., Erlank, A., Groenewald, C. and Barnard, A.: The OuterNet: A novel satellite communication relay constellation. In: Sandau, R., Kawashima, R., Nakasuka, S. and Sellers, J.J. (eds.), *Innovative Ideas for Micro/Nano-Satellite Missions*, pp. 63–75. International Academy of Astronautics, France, 2013.

[2] Klofas, B. and Leveque, K.: The Future of CubeSat Data Communications (Presentation). In: *AMSAT-NA Space Symposium*. Orlando, Florida, 2012.

[3] Wertz, J.R. and Larson, W.J.: *Space Mission Analysis and Design*. Microcosm Press, 1999.

[4] Grini, D.: RF Basics, RF for Non-RF Engineers. In: *MSP430 Advanced Technical Conference*. 2006.

[5] Circuit Design, Inc.: RF Design Guide. Accessed: 2013.
Available at: `http://www.cdt21.com/resources/guide3.asp#Types%20of%20antenna`

[6] International Telecommunications Union, Radiocommunication Sector of ITU: Recommendation ITU-R P.676-10 (09/2013), Attenuation by Atmospheric Gases. 2013.

[7] Maral, G. and Bousquet, M.: *Satellite Communication Systems: Systems, Techniques and Technology*. 5th edn. Wiley, 2009.

[8] Bezuidenhout, Q.: *Strategy Selection for Optimal LEO Satellite Communication*. Masters Thesis, Stellenbosch University, December 2012.

[9] Jones, G.: *Packet Radio: What? Why? How?* TAPR, 1995.

[10] Beech, W.A., Nielsen, D.E. and Taylor, J.: *AX.25 Link Access Protocol for Amateur Packet Radio v2.2*. Tucson Amateur Packet Radio Corporation (TAPR), 1998.

[11] McGuire, J., Galysh, I., Doherty, K., Heidt, H. and Neimi, D.: *FX.25: Forward Error Correction Extension to AX.25 Link Protocol For Amateur Packet Radio*. Stensat Group, 2006.

[12] Klofas, B.: CubeSat Communications System Table. Accessed: August 2014.
Available at: `http://www.klofas.com/comm-table/`

[13] Han, Y.S.: BCH Codes, Graduate Institute of Communication Engineering, National Taipei University, Taiwan. Accessed: 2014.
Available at: `web.ntpu.edu.tw/~yshan/BCH_code.pdf`

[14] Klofas, B., Anderson, J. and Leveque, K.: A Survey of CubeSat Communication Systems. In: *CubeSat Developers' Conference*. California Polytechnic State University and SRI International, November 2008.

[15] Mehrparvar, A.: *CubeSat Design Specification, Rev 13*. California Polytechnic State University, February 2014.

[16] Xiantai, G., Hongchao, Y., Feng, Y. and Gang, L.: Modelling and Simulation of Small Satellite Constellation Networking Using Multi-topology Routing. In: *International Conference on Computer Application and System Modelling*, vol. 12, pp. V12–143–V12–147. 2010.

[17] Muri, P., McNair, J., Antoon, J., Gordon-Ross, A., Cason, K. and Fitz-Coy, N.: Topology Design and Performance Analysis for Networked Earth Observing Small Satellites. In: *Military Communications Conference (MILCOM)*, pp. 1940–1945. November 2011.

[18] Houyou, A.M., Holzer, R., Meer, H.D. and Heindl, M.: Performance of Transport Layer Protocols in LEO Pico-Satellite Constellations. Tech. Rep., Faculty of Computer Science and Mathematics, University of Passau, December 2005.

[19] Hogie, K. and Parise, R.: Using Standard Internet Protocols and Applications in Space. *Computer Networks*, vol. 47, pp. 603–650, 2005.

[20] Neglia, G., Mancuso, V., Saitta, F. and Tinnirello, I.: A Simulation Study Of TCP Performance Over Satellite Channels. In: *34th COSPAR Scientific Assembly*. Houston, 2002.

[21] Boussemart, V. and Brandt, H.: A Tool for Satellite Communications: Advanced DVB-RCS / DVB-S2 System and Protocol Simulator. In: *Signal Processing for Space Communications*, pp. 1–5. October 2008.

[22] van Staden, T.: *Investigation Into the Optimization of Low Speed Communication Protocols for Narrow Band Networks*. Masters Thesis, University of Stellenbosch, December 2013.

[23] Henderson, T.R. and Katz, R.H.: Transport Protocols for Internet-Compatible Satellite Network. *IEEE Journal On Selected Areas Of Communications*, vol. 17, pp. 326–344, 1999.

[24] Kruse, H.: Optimizing Data Transmission Capacity on Satellite Links with Non-Zero Bit Error Rates. Tech. Rep., McClure School of Communication Systems Management.

[25] Henderson, T.R. and Katz, R.H.: A Link Layer Protocol For Efficient Transmission of TCP/IP via Satellite. In: *MILCOM*, vol. 2, pp. 723–727. November 1997.

[26] Dubroca, S.: *Cross-Layer Optimization In A Satellite Communication Network*. Masters Thesis, KTH Royal Institute of Technology, August 2013.

[27] ns2 Network Simulator Home Page. Accessed: June 2013.
Available at: `http://nsnam.isi.edu/nsnam/index.php/Main_Page`

[28] ns3 Network Simulator Home Page. Accessed: June 2013.
Available at: `http://www.nsnam.org/`

[29] Riverbed Technology: OPNET Modeller. Accessed: April 2014.
Available at: `http://www.riverbed.com/products/`

[30] PyEphem. Accessed: January 2014.
Available at: `https://pypi.python.org/pypi/pyephem`

[31] Clear Sky Institute: XEphem. Accessed: January 2014.
Available at: `http://www.clearskyinstitute.com/xephem/`

[32] SimPy. Accessed: January 2014.
Available at: `http://simpy.readthedocs.org/en/latest/`

[33] Matplotlib Basemap Toolkit. Accessed: January 2014.
Available at: `http://matplotlib.org/basemap/index.html`

[34] Panda3D. Accessed: January 2014.
Available at: `https://www.panda3d.org/`

[35] Sphinx Python Documentation Generator. Accessed: March 2014.
Available at: `http://sphinx-doc.org/`

[36] Friis, H.: A Note on a Simple Transmission Formula. *Proceedings of the IRE*, vol. 34, no. 5, pp. 254–256, May 1946.

[37] Lathi, B.P. and Ding, Z.: *Modern Digital and Analogue Communication Systems*. 4th edn. Oxford University Press, 2010.

[38] Wesdock, J., Patel, C., Bustamante, E., Michel, G., Zhang, Y., Wheeler, D. and Canzona, N.: *Communications Receiver Performance Degradation Handbook*. Joint Spectrum Center, Annapolis, Maryland, August 2010.

[39] Dhotre, I. and Bagad, V.: *Computer Communication Networks*. Technical Publications, 2006.

[40] Sadikoglu, F.: Digital Modulation Techniques in Mobile Communications, Near East University, Cyprus. Lecture Presentation.

[41] Nelson, D.R.A.: Satellite Communication Systems Engineering: LEO, MEO, GEO. Satellite Engineering Training Course, 2001.

[42] Freeman, R.L.: *Fundamentals Of Telecommunications*. Wiley, 1999.

[43] FEKO Homepage.
Available at: `http://www.feko.info`

[44] Ippolito, L.J.: *Satellite Communications Systems Engineering*. Wiley, 2008.

[45] Sandau, R., Nakasuka, S., Kawashima, R. and Sellers, J.J. (eds.): *Innovative Ideas For Micro/Nano-Satellite Missions - Volume 1, Number 3*. The International Academy of Astronautics, 2013.

[46] Sandau, R., Nakasuka, S., Kawashima, R. and Sellers, J.J. (eds.): *Novel Ideas For Nanosatellite Constellation Missions - Volume 1, Number 1*. The International Academy of Astronautics, 2012.

[47] Katona, Z. and Berioli, M.: Design of Circular Orbit Satellite Link for Maximum Data Transfer. In: *IEEE International Conference on Communications (ICC)*, pp. 1–6. June 2011.

[48] International Telecommunications Union, Radiocommunication Sector of ITU: Recommendation ITU-R P.835-5 (02/2012), Reference Standard Atmospheres. 2012.

[49] MacKay, D.: *Information Theory, Interference, and Learning Algorithms*. 4th edn. Cambridge University Press, March 2005.

[50] Lertpratchya, D., Riley, G.F. and Blough, D.M.: Simulating Frame-Level Bursty Links in Wireless Networks. In: *7th International ICST Conference on Simulation Tools and Techniques*, pp. 108–117. Brussels, Belgium, March 2014.

[51] Zimmermann, H.: OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, Apr 1980.

[52] Grønstad, M.A.: *Implementation of a Communication Protocol for CubeSTAR*. Masters Thesis, University of Oslo, July 2010.

[53] Riley, M. and Richardson, I.: An Introduction to Reed-Solomon Codes: Principles, Architecture and Implementation, Carnegie Mellon University. Accessed: July 2014.
Available at: `http://www.cs.cmu.edu/~guyb/realworld/reedsolomon/reed_solomon_codes.html`

[54] Stensat Group: FX.25 Working Example Source Code. Accessed: July 2014.
Available at: `http://www.stensat.org/projects/FX-25/FX-25_performance.htm`